NTNU – Trondheim
Norwegian University of
Science and Technology

Analysis of key reuse attacks on the Post-Quantum-scheme NewHope

Markus Swarowsky

Submission date: April 2020
Supervisor: Colin Boyd, IIK
Co-supervisor: Bor de Kock, IIK

NTNU – Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title:	Analysis of key reuse attacks on the Post-Quantum-scheme NewHope
Student:	Markus Swarowsky

Problem description:

In the thesis the current attack from Bauer et al. will be analysed. This will be done by re-implementing the attack with an only CPA-secure fork of the official reference implementation. Additionally the optimization from Qin et. al. will be added. Afterwards it can be evaluated whether the same results are achieved as published in the two papers. As there is a huge difference between the number of queries the two paper uses, it will be investigated if it is possible to combine the optimizations from Qin et al. with the base attack from Bauer et. al. Also the Bauer paper suggests that there might be further optimizations. Another open question is if there can be a key mismatch oracle build out of the CCA-secure version of NewHope.

Date approved:	2019-11-14
Supervisor:	Colin Boyd, IIK

Abstract

Up to now, quantum computers have only been considered a theoretical threat to today's public-key-cryptography. Also, nobody can say exactly how long it will take until the first larger quantum computers exist. But recently some progress has been made, so this theoretical threat is slowly turning into a real one. Therefore, the cryptographic research community has started to develop new schemes that are also safe against attacks of quantum computers.

One of these new schemes is NewHope from Alkim et al. [2], but as with all new schemes the best way to build confidence that a scheme is as secure as claimed by the authors is to have it analyzed by the research community. This work is part of this analysis as we take a look at the so far published attack ideas from Bauer et al. [4] and Qin et al. [36]. These are key reuse attacks, which target the passively secure version of NewHope. We re-implemented the attacks and tested them against the C reference implementation written by the authors of NewHope. Within this process it was possible to identify minor and major problems. While the Problem in the approach from Bauer et al. just caused a lower success rate, the improvement from Qin et al. became infeasible. Therefore we developed other improvements, to speed up the attack but also to make it possible to recover over 99% of the secret keys.

Preface

In my master studies I took several courses ranging from the formal principles of cryptography up to their application. Although I now know and can explain many of the basics, it still fascinates me and has lost none of its "magic". For example, the Diffie-Hellmann key exchange, which only allows to exchange a secret key by clever use of mathematics without the possibility for an eavesdropper to obtain it. For this reason I wanted for my thesis to use my gained theoretical knowledge to once dive into a practical cryptoanalysis and what specific field could have been more interesting as new upcoming post-quantum cryptography.

Therefore I am super grateful to all people at the Karlsruhe Institute of Technology(KIT) and at the Norwegian University of Science and Technology(NTNU) to make this thesis possible. In special Willi Geisselman, to support me with all the bureaucracy. but also my supervisors Colin Boyd and Bor de Kock for answering all my questions and their helpful remarks during the thesis. I would also like to thanks my parents and all my friends that supported me during the embarrassing and dark times during the thesis as well as all the volunteers who cross-read the thesis to identify issues. Finally shout out to the high-class music of Scooter, which always brought me back on track.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

CCA Chosen-cipher text attack.

CPA Chosen-plain text attack.

CVP Closest Vector Problem.

KEM Key Encapsulation mechanism.

LWE Learning with Errors.

NIST National Institute of Standards and Technology.

NTT Number Theoretic Transform.

PKC public key cryptography.

PQ post-quantum.

RLWE Ring-Learning with Errors.

SVP Shortest Vector Problem.

Notation

$\lfloor x \rceil$	if $x \in \mathbb{R}$ $\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor \in \mathbb{Z}$
$e \xleftarrow{R} \mathcal{D}$	sample a uniformly random element e from the set or distribution $\mathcal{D}$
$\mathcal{L}(B)$	a lattice create by the basis B
$v[i]$	the i-th element of a vector v or the coefficient of x^i of a polynomial v
$\mathbb{Z}_q$	$\mathbb{Z}/q\mathbb{Z}$, the integer remainder set mod q with the elements $[0, q-1]$
$\mathcal{R}_q$	$\mathbb{Z}_q[x]/(x^N+1)$, with q being a prim number an $N \in \mathbb{Z}$
$\hat{\mathbf{c}}$	polynomial in $\mathbb{Z}_8[x]/(x^N+1)$, a compressed element in NewHope
$Sign(0) > 0$	zero will be denote as positive
$\lvert v \rvert$	if v is a string, length of a string
ν	the key string that is used the derive the final shared secret key
l	the quadruplet (l_0, l_1, l_2, l_3) of cipher text variations for one bit of ν
sum v	internal sum of the key mismatch oracle 4.5 $v = \sum_{j=1}^{3} \lvert l_j - s_j \rvert - 8$
ψ_k	centered binomial distribution with parameter $k > 0$ To get a sample from ψ_k compute $s = \sum_{i=1}^{k} b_i - b'_i : b_i, b'_i \in \{0,1\}$
$\mathbf{a}$	polynomials will be denoted as bold $a[x] \hat{=} \mathbf{a}$ This notation is inherited from the Learning with Errors (LWE) field
s_0, s_1, s_2, s_3	the four secret coefficients involved in decoding one bit of ν $(\mathbf{s}[i], s[i+256], \mathbf{s}[i+512], \mathbf{s}[i+768]) : i \in [0, 255]$
$f_v(l_j)$	function of key mismatch oracle output of the input of l_j $f_v(l_j) = \lvert l_j - s_j \rvert + v : j \in [0, 3]$
$Sign(x)$	outputs a 0 for all $x \geq 0$ and 1 otherwise. In the context of the key mismatch oracle $(0, 1)$ gets mapped to $(+, -)$

Chapter 1
Introduction

Ever since humans have been exchanging information, there has been the demand to do this confidentially. To achieve this confidentiality, cryptographic schemes were used early on to encrypt information. The way these cryptographic schemes work has changed over time. It is a normal process that schemes that once were considered to be secure can later easily be broken, due to technical innovations and research. An early example of this is the Vigenère cipher, which dates back to the 16th century and was called «le chiffre indéchiffrable» (French for «the indecipherable cipher») for a long time [29]. Due to the increased emergence of the frequency analysis, it could be broken in the 19th-century [34]. With today's technology, the attack is considered trivial and the cipher is far away from being a «indecipherable cipher». This process of developing new schemes and breaking them continues until today, where the Internet allows us to exchange more information than ever before.

Therefore cryptography has become even more important. Modern secure internet protocols, like TLS, use several types of cryptography to not only provide confidentiality, but also other important properties such as authenticity and integrity. However, it seems that we are once again on the verge of time where new technology makes existing cryptography insecure. This time it will be quantum computers that create the possibility to break parts of the cryptography currently used. So far they have not been considered as a threat, although the idea of quantum computers has been around for a few decades and special algorithms have already been designed for them. However, it seems that a practical realization is potentially close, as meanwhile some major parties like, the NSA, Google, IBM and the Chinese government are working on building a real usable quantum computer. This would make it possible to solve certain mathematical problems quickly and efficiently, but would also eliminate the security of the today's used public key cryptography. For this reason, new schemes must be developed that remain secure even if there are attackers with access to a quantum computer.

To standardize such a new post-quantum scheme, the National Institute of Standards and Technology (NIST) makes use of a proven procedure and has started a competition in 2016. This competition involves researchers submitting new schemes to then allow everybody to analyse and evaluate them. If a scheme turns out to be impractical or insecure, it will be discarded. New technologies on the attacker side enforce that also new concepts in cryptography are needed to withstand these new attackers. Most of the 69 schemes submitted in round one of the NIST-competition can be divided into four categories:

- Lattice-based
- Code-based
- Isogeny-based
- Multivariate-based

This thesis focus on the scheme NewHope developed by Alkim et al. [2] that is based on the mathematical principle of lattices, which comes with the the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP). These are the computational hard to solve problems that are required to build public key cryptography. A special way to use these problems to create cryptographic schemes is the Learning with Errors (LWE) approach, which uses matrices as a basis for the lattices and thus also for the keys. Therefore the Ring-Learning with Errors (RLWE) principle was developed which is based on LWE idea but uses polynomials in a ring instead of matrices and thus requires less computing power. One promising concrete implementation of RLWE is the scheme NewHope. It was already used by Google, where they added the NewHope encryption on top of the existing one, to test how post-quantum cryptography could be rolled out on the Internet [9].

As the lattice-based approach is way more complicated than the currently used cryptography a lot of analysis is needed to gain confidence in the new schemes. This thesis is a part of this process by analyzing the possibilities of a key mismatch attack on NewHope, by achieving the following goals:

- Understanding the NewHope scheme
- Analyse the published key reuse attack by re-implementing and testing it against the reference implementation
- Discover a minor problem in the basic attack from Bauer et al.
- Discover two major problems in the improved attack form Qin et al.
- Develop own improvements of the attack

The thesis will start with a short introduction on what quantum computers are, including why they can break certain forms of cryptography. To then explain what lattices are and how it can be uses to create PQ cryptography with based on the the SVP and the CVP. Followed by a description of the Ring-Learning-with-Errors principle that is used in NewHope.

With these foundations, we envision the first basic attack published by Bauer et al. [4], which focuses on an attack against the server party of the scheme and allows it to find out major parts of its secret server key. It is important to note that the attack is not against the final version of NewHope, but only against the passively secure version. Additionally, it is also assumed that the key parameters are reused over time. To analyze the attack, it is re-implemented and tested against the reference implementation, written by the authors of NewHope. During this analysis, we discovered a minor problem with the attack, which slightly reduces the success rate. Building on this basic attack, Qin et al. published an improvement that should make it possible to recover almost the entire secret key [36]. We also re-implemented this improvement and tested it against the reference implementation. This test revealed two problems which lead to the fact that the improvement cannot be executed successfully. For this reason, we have developed improvements that enable us to recover almost the entire secret key and speed up the attack. Therefore making it more practical and a bit more considerable.

Chapter 2
Preliminaries

The second chapter gives a small explanation of public key cryptography, KEMs, quantum computers and why they are a threat for today's cryptography, but also what is done to face that threat.

2.1 Public Key Cryptography

Public key cryptography (PKC) is a subfield of cryptography, where the communicating participants have two associated keys, a private and a public one. The public key should be made available to the public. Thus it can be used to encrypt messages, which can only be decrypted by the owner of the corresponding private key. Therefore the private key should only be made available to the parties who are supposed to read the messages.

Definition 2.1. Security parameter *A security parameter lambda indicates how complicate it is for adversary to break a instance of a scheme instantiated with it. Often it is related to the key size.*

Definition 2.2. Polynomial time algorithm *If the run time of an algorithm is within $O(n^k)$ for its input size n and any $k > 0$, it is a polynomial time algorithm.*

Definition 2.3. Public Key scheme
Let $\mathcal{M}$ be the message space, let λ be a security parameter. A public key scheme consists of three polynomial time algorithms:

- $(pk, sk) \leftarrow$ *Key Generation*(1^λ) : outputs a public and private key pair
- $ctxt \leftarrow$ *Encrypt*(m, pk) : encrypts a message $m \in \mathcal{M}$, with the public key pk
- $m' \leftarrow$ *Decrypt*$(ctxt, sk)$: decrypts a ciphertext $ctxt$ to the message m' with the private key sk

When public key cryptography was introduced by Diffie and Helllman in the 1970's [12] it opened a wide variation of new possibilities. In contrast to symmetric cryptography, where the same key is used for encryption and decryption, confidential communication between two parties is now possible without the need of a common shared secret. The security of the schemes is usually based on mathematical problems that are difficult to solve, such as the factorization of large integers [22], which for example is used in the RSA scheme, which was published in 1978 by Rivest, Shamir and Adleman [38].

Definition 2.4. integer factorisation problem
Let $N = q \times p, q \neq p$ and p, q prime numbers.
Problem: Given N, find p and q

2.1.1 Key Encapsulation Mechanisms

Due to underlying mathematics public key cryptography requires more computing power than symmetric cryptography. Therefore hybrid encryption schemes were designed. The idea of them is to use the techniques of public key cryptography to exchange a symmetric key, which then is used for further communication. This way the advantages of both systems are combined, it is possible to communicate securely without a shared secret and to work efficiently with resources.

Definition 2.5. Key Encapsulation mechanism (KEM)
Let $\mathcal{K}$ be the key-space of an associated symmetric key scheme, let λ be a security parameter. A KEM consists of three polynomial time algorithms:

- $(pk, sk) \leftarrow$ *Key Generation*(1^λ): outputs a public and private key pair
- $(k, ctxt) \leftarrow$ *Encapsulation*(pk): outputs a ciphertext and shared secret key $k \in \mathcal{K}$
- $k' \leftarrow$ *Decapsulation*$(ctxt, sk)$: outputs the decapsulated shared secret key k' from the ciphertext ctxt
- The two keys should be equal $(k = k')$

One of the main differences between PKC and KEMs is that *Encrypt* allows the message that it will encrypt to be specified by the sender. *Encapsulation* on the other hand selects internally a random value that is encrypted. Also the distribution of the public key is different. In public key schemes, the key is sent to the communication partners once and used on a long-term basis. While the protocol procedure of KEMs usually foresees that the public key is sent each time. This is based on the assumption that KEMs are mostly used for a key exchange between two parties that have not communicated with each other before.

A KEM that is widely used today is the Diffie-Hellman KEM, which was published by Diffie and Hellman in 1979 [12]. Its security is based on the discrete logarithm problem.

Definition 2.6. Discrete logarithm problem
Let b an element of the group $\mathbb{G}$.
Problem: Given $a = b^x$, find $x = \log_b a$

Example 2.7. Diffie-Hellman KEM
Let p be a prime number, then $\mathbb{G}_p$ is multiplicative group of integers modulo p, with $p-1$ elements. An element $g \in \mathbb{G}_p$ is called generator if all elements of $\mathbb{G}_p$ can created by multiplying g with itself.

- *Key Generation*(1^λ):
  Choose a prime p; a generator $g \in \mathbb{G}_p$ and $a \xleftarrow{R} \text{ord}(g) - 1$
  Output: $pk = (pk' = g^a \mod p, g, p)$ and $sk = a$
- *Encapsulation*$(pk = (pk', g, p))$:
  Choose $b \xleftarrow{R} \text{ord}(g) - 1$
  Output: $k = pk^b = (g^a)^b = g^{ab} \mod p$ and $ctxt = g^b \mod p$
- *Decapsulation*$(ctxt, sk)$:
  Output: $k' = ctxt^a = (g^b)^a = g^{ab} \mod p$

2.1.2 Security models of KEMs

To determine the security of a KEM the indistinguishability game (IND) is used. In this game an attacker A will get a ciphertext $ctxt_b$ and a key k_b that is either a random key or the actual encapsulated key of $ctxt_b$. If the attacker can determine the correlation between $ctxt_b$ and k_b correctly he wins the game. The run time of A must be polynomial relative to the security parameter.

For the level of security it depends on how powerful the attacker is, which is defined by the attack model. The two relevant ones for this work are the Chosen-plain text attack (CPA) and the Chosen-cipher text attack (CCA). In the CPA model the attacker gets the public key and can use the *Encapsulation* algorithm to gather information that will help him to win the IND game. In the CCA model the attacker has additionally access to a decapsulation oracle *Dec*, which allows him to create own ciphertexts and get the key that will be decapsulated out of them. This gives the attacker more power, which is why a scheme that withstands a CCA attacker is considered to be more secure than one that only withstands an CPA-attacker.

Together the combination of the game and the attacker model form the full security definition of a scheme. Figure 2.1 illustrates the IND-CPA game and Figure 2.2

shows the IND-CCA game for KEMs with the additional decapsulation oracle *Dec*. In both games the attacker wins if $b = b'$.

1. $(pk, sk) \leftarrow$ *Key Generation*(1^λ)
2. $b \xleftarrow{R} \{0,1\}$
3. $k_0 \xleftarrow{R} \mathcal{K}, ctxt_0 \xleftarrow{R} \{0,1\}^*$
4. $(k_1, ctxt_1) \leftarrow$ *Encapsulation*(pk)
5. $b' \leftarrow A(pk, ctxt_b, k_b)$
6. Output : $b \stackrel{?}{=} b'$

Figure 2.1: $\text{Game}_{\text{IND-CPA}}$ for KEMs by Bellare et al. [5]

1. $(pk, sk) \leftarrow$ *Key Generation*(1^λ)
2. $b \xleftarrow{R} \{0,1\}$
3. $k_0 \xleftarrow{R} \mathcal{K}, ctxt_0 \xleftarrow{R} \{0,1\}^*$
4. $(k_1, ctxt_1) \leftarrow$ *Encapsulation*(pk)
5. $b' \leftarrow A_{Dec}(pk, ctxt_b, k_b)$ and $ctxt_b \notin S$
6. Output : $\begin{cases} 1 & \text{if } b = b' \\ 0 & \text{otherwise} \end{cases}$

Algorithm 2.1 Oracle Dec

Input: $ctxt$

$S \leftarrow S \cup \{ctxt\}$

return Decapsulation$(ctxt, sk)$

Figure 2.2: $\text{Game}_{\text{IND-CCA}}$ for KEMs

Since the probability to win the game by guessing is a half, a scheme is only considered as broken if an attacker wins the game significantly more often than that, as shown in Definition 2.9

Definition 2.8. Negligible function *A function ϵ is negligible if it approaches zero faster than the reciprocal of every polynomial:*

$$\epsilon \text{ negligible} \iff \forall c \in \mathbb{N}, \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : \epsilon < \frac{1}{n^c}$$

Definition 2.9. (IND-CPA/CCA) security *A key encapsulation mechanism is IND-CPA/CCA secure if the advantage of every polynomial time attacker over a random guess is negligible in the security parameter, formally:*

$$Adv^{\text{IND-CPA}}(A) := \left| \Pr[1 \leftarrow Game_{IND\text{-}CPA}(1^\lambda)] - \frac{1}{2} \right| \leq negl(\lambda)$$

Analog for IND-CCA security.

$$Adv^{\text{IND-CCA}}(A) := \left|\Pr[1 \leftarrow Game_{IND\text{-}CCA}(1^{\lambda})] - \frac{1}{2}\right| \leq negl(\lambda)$$

2.2 Quantum Computing

The main idea of a quantum computer is to apply the properties of quantum physics to information processing. But so far it is an open research question whether and how the theoretical construct of a quantum computer can be realized in a large and useful scale. At present, there are several smaller quantum computers, but intensive research is underway to develop larger versions. In 2019 Google claimed to have built the first quantum computer superior to a classical computer and thus declared quantum supremacy [27] [35]. Although the results are questioned by the research community [31], there is a large community that thinks that great progress is being made towards more and more powerful quantum computers.
Based on the Preskill lecture notes [35], Nielsen & Chuang [30] and Tiepelt [45], this section will give a short and simplified introduction what quantum computers are and why they are a threat for the currently used cryptography.

2.2.1 Qubits

The fundamental differences between a quantum computers and classical computers is the way information is stored and processed. While classical computers use bits, quantum computers use quantum bits, so-called qubits. A classical bit can only have exactly one fixed state at a time, which is usually described as zero or one. In addition to these two states, a qubit can also have a superposition of zero and one. These three states can be represented as vector:

$$\text{zero: } |0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \quad \text{one: } |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$$

$$\text{superposition of zero and one } |\psi\rangle : \alpha|0\rangle + \beta|\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix} : \alpha, \beta \in \mathbb{C}$$

To determine the state of a qubit that is in a superposition $|\psi\rangle$ a measurement has to be done. This then will force it to either take state $|0\rangle$ or $|1\rangle$ and the superposition will collapse. Which state it takes depends on amplitudes α and β, by taking the state $|0\rangle$ with probability $|\alpha|^2$ and the state $|1\rangle$ with probability $|\beta|^2$.

To use more than one qubit a n-qubit big quantum register is used, which can be represented by a 2^n-dimensional vector. But unlike registers in a conventional

computer, the qubits in a quantum register are not independent. So the register can be a superposition of 2^n different states, which is often denoted as the sum $\sum_{x=0}^{2^n-1} \alpha_x|x\rangle$.

For a 2 qubit register this means there are the two vector spaces V_1 and V_2 with the basis $|0\rangle$ and $|1\rangle$. Their state can be described as $\alpha_i|0\rangle + \beta_i|1\rangle$, then the register will contain elements in the vector space $V_1 \otimes V_2$, which can be expressed in the form:

$$\begin{aligned}&(\alpha_1|0\rangle + \beta_1|1\rangle) \otimes (\alpha_2|0\rangle + \beta_2|1\rangle)\\ =&\alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \alpha_2\beta_1|10\rangle + \beta_1\beta_2|11\rangle\end{aligned}$$

If a measurement of this register is performed the superposition collapses and the measured state depends on the probabilities of the amplitudes:

$$\begin{aligned}|\alpha_1\alpha_2|^2 \to 00; \qquad |\alpha_1\beta_2|^2 \to 01\\ |\alpha_2\beta_1|^2 \to 10; \qquad |\beta_1\beta_2|^2 \to 11\end{aligned}$$

One of the major tasks in building a quantum computer is to create a quantum register as large as possible. The number of qubits is also an indication of the performance, since more qubits can be used to represent more states in parallel. When referring to large or powerful quantum computers, register sizes of several hundred to thousand qubits are intended. The currently largest published interim result is from Google AI Quantum, which managed to produce 53 qubits in 2019 [27].

With a working quantum register in place, simple operations can be done on single or multiple qubits. This can be modeled by quantum circuits, which try to follow the same principle as classical digital circuits. So there is a number of input registers, which then are connected to gates that perform an action on qubits and might change their state based on their input state. The outputs can than be again used as new input for further gates. These quantum gates can do operations that are also known from classical circuits, like a NOT-gate that negates the input qubit, but there are also special quantum gates, like the Hadamard gate, which creates a equally distributed superposition of $|0\rangle$ and $|1\rangle$.

2.2.2 Problems for Cryptography

By creating an input set of qubits, then applying a series of operation and measuring them afterwards it is possible to do computation. As an n-set of qubits can have a superposition of 2^n states at the same time, the computation can be performed for 2^n states at once. Special quantum algorithms combine all these states in a way that all irrelevant states destructively interfere with each other. Thus, the superposition collapses into the best state when a measurement is made after the algorithm has

been executed. This feature enables quantum computers to solve particular problems much more efficiently than classical computers, like the factorization of large integers mentioned in definition 2.4. The best known algorithm to solve this problem with a classical computer is the general number field sieve, which needs

$$O\left(e^{\frac{64}{9}n^{\frac{1}{3}}(\log n)^{\frac{2}{3}}}\right)$$

operations, with n bits to represent the number [19]. This makes solving it infeasible for numbers greater than 1024 bits with today's computer hardware. The last published record from Thomé et al. is 829 bit [14], which was achieved on a high performance cluster in 2020. But already in 1994, Shor published a quantum algorithm to factorize integers [39], which has a logarithmic run time that can be expressed as:

$$O\left((\log n)^2 \cdot \log\log n\right)$$

So with Shor's algorithm in place, the factoring problem can be solved for way bigger integers, as soon as a quantum computer with enough qubits is there. As the security of the RSA encryption [38] is based on the integer factoring problem, it can be directly broken with a quantum computer, with the normally used key sizes of 2048 to 4096 bits. Considered that RSA is commonly used today it becomes clear that quantum computers are a threat to the public key cryptography currently used. Besides the factoring problem also the discrete logarithm can be computed more efficiently. Hence the Diffie–Hellman key exchange described in Example 2.7 will also be broken.

2.3 NIST-Competition

As mentioned before, the threat of big and working quantum computers starts to grow. So the National Institute of Standards and Technology raised a competition near the end of 2016, for Public-Key post-quantum (PQ) cryptographic algorithms [10]. The aim is to find two new types of PQ algorithms, first public-key encryption and KEM algorithms and second digital signature algorithms. The KEM candidates were demanded to fulfill certain requirements, which include:

- IND-CCA secure
- breaking it should need at least the same amount of resources as breaking AES-256 [11]
- an attacker is allowed to use up to 2^{64} decapsulation queries,
- a quantum attacker is limited to a quantum circuit depth of 2^{64} which is aprox. equivalent to 2^{234} of classical computation operations.

The reason why the competition was already started, even though it may take several decades before a large quantum computer exists, is that if somebody already records messages at the moment and stores it, the content could be decrypted later when they are available. This is why the KEM protocols should be replaced before that is the case. On the other hand, authentication only needs to be secure at the moment the protocol is executed. Therefore the authentication protocols can be replaced later.

Also the actual communication that is encrypted with an symmetric encryption scheme is less affected, as symmetric encryption like AES [11] is in principal secure against quantum computer attacks if the key size is increased [6]. Increasing the key size is also an option for current public key cryptography like RSA [38], which led to submission of post-quantum RSA [7] to the competition. But while the larger AES keys are still practical to use the PQ-RSA keys are approx. 1-terabyte big [7], which would not be practical with current computers. So new schemes have to be designed and analysed, which is why National Institute of Standards and Technology (NIST) has already started the NIST-competition in 2016.

In the first round 69 submissions where accepted. After one year of evaluation by the international crypto-community the candidates of round two where announced on the 30th of January 2019. All broken candidates were removed and some candidates merged, which resulted in a list of 26 submissions that are studied and analysed at the moment. Including the NewHope scheme, which is focused in this work and described in section 3.3. So the number of submissions that will be approved for round three in 2020 can be reduced further. In this way it should be ensured that only the best and sufficiently investigated procedures reach the final round. This also helps to build confidence that there will be no security problems in the final candidates. A similar process was used to define the hash function SHA-3 [8] and the symmetric scheme AES [11].

Chapter 3
Lattice-based cryptography

This chapter explains what lattices are and how they can be used to create a cryptographic scheme. This is demonstrated by the Ring-Learning with Errors (RLWE) approach and is based on Micciancio and Regev [28] and Peikert [32]. At the end the description of the RLWE-based scheme NewHope will be presented.

3.1 Lattice

Definition 3.1. Integer Lattice *An integer lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^n$ that is spanned by a linear integer combination of a basis of $\mathbb{R}^n$.*
Let $B = (\vec{b_1}, \vec{b_2}, \ldots, \vec{b_n})$ be an integer basis. Then $\mathcal{L} = \{\sum_{i=1}^{n} z_i \vec{b_i} : z_i \in \mathbb{Z}, \vec{b_i} \in B\}$

According to Definition 3.1 the most general lattice is the subgroup $\mathbb{Z}^n \subset \mathbb{R}^n$. In the rest of this work we will only consider integer lattices $\mathcal{L} \subseteq \mathbb{Z}^n$. As a lattice is a discrete group it can be seen as a finite set of points that all can be represented by a linear combination of the basis vectors $(\vec{b_1}, \vec{b_2}, \ldots, \vec{b_n})$ of the basis B, which are linearly independent. It should be noted that the basis is not unique. If the base is represented as a matrix and the individual base vectors are considered as rows of the matrix B, a unimodular matrix U can be found for which the following applies:

$$UB = B' \text{ and } B = U'B'$$
$$\text{from which follows: } B = U'B' = U'UB \implies U'U = I \implies U' = U^{-1}$$

Therefore B and B' are both a valid basis of the same lattice, while U just represents the linear combination of the basis vectors of B to represent the vectors of B'. However, there are preferred bases because some problems over lattices can be solved more efficient with them, details will follow with Definition 3.5 and 3.6. Such a «good» basis is characterized by the fact that its base vectors are short and pairwise as orthogonal as possible to each other. Figure 3.1 illustrates an examples of a lattice with a «good» and a «bad» basis.

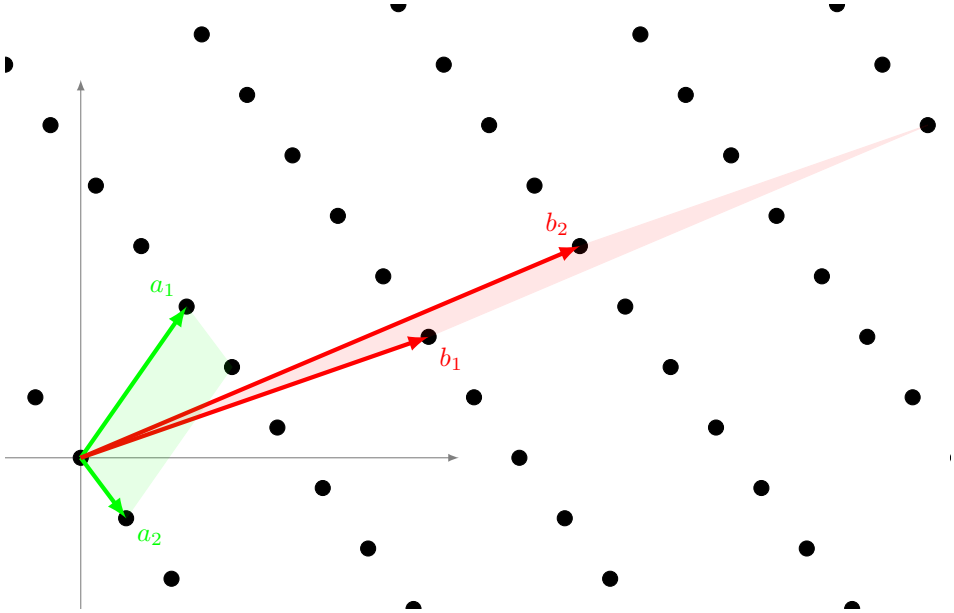


Figure 3.1: A lattice with a «good» $A = (\vec{a_1}, \vec{a_2})$ and a «bad» $B = (\vec{b_1}, \vec{b_2})$ basis.

Definition 3.2. Successive Minima
Let $\mathcal{L}$ be a lattice and $\|v\|$ the euclidean norm of v.

- $\lambda_1(\mathcal{L}) = \min_{0 \neq v \in \mathcal{L}}(\|v\|) = \min_{x \neq y \in \mathcal{L}}(\|x - y\|)$
- $\lambda_i(\mathcal{L}) = \lambda_{i-1} \cup \min(\|v\| : \forall v : \|v\| \geq \|v'\|), v \in \mathcal{L}, v' \in \lambda_{i-1}$

The successive minima (Definition 3.2) gives an indication of size of the vectors $(\vec{v_1}, \vec{v_2}, \dots) \in \mathcal{L}$ of the lattice. By sorting these values in ascending order, so $\lambda_1(\mathcal{L}) \leq \lambda_2(\mathcal{L}) \leq \lambda_3(\mathcal{L}) \leq \cdots \leq \lambda_n(\mathcal{L})$ means $\|\vec{v_1'}\| \leq \|\vec{v_2'}\| \leq \|\vec{v_3'}\| \leq \cdots \leq \|\vec{v_n'}\|$. In lower dimensions ≤ 4 the vectors $\vec{v_1'}, \vec{v_2'}, \vec{v_3'}, \vec{v_4'}$ will always form a basis, but this is not the case for higher dimensions (≥ 5) [26].

Definition 3.3. Lattice determinant
Let $\mathcal{L}$ be a lattice. The determinant $\det(\mathcal{L})$ of the lattice $\mathcal{L}$ is defined as:

- $\det(\mathcal{L}) = \sqrt{\det(B^T B)}$, *where B is any basis for $\mathcal{L}$*
- *or* $\prod_{i=1}^{n} \|\vec{b_i}\|$, *where $\vec{b_i}, i \in \{1, 2, \dots, n\}$, are basis vectors*

Theorem 3.4. ***Minkowski***
For every integer $r > 1$, there exists constant γ_r, such that for any lattice $\mathcal{L}$ of rank

r and for all $1 \leq k \leq r$*:*

$$\left(\prod_{i=1}^{k} \lambda_i(\mathcal{L})\right)^{\frac{1}{k}} \leq \sqrt{\gamma_r}\det(\mathcal{L})^{\frac{1}{r}}$$

The Minkowski theorem gives a relation between the lattice determinant defined in definition 3.3 and the successive minima. Therefore, an estimation of the successive minima can be given, just by knowing a basis. This can be helpful for solving some computational hard problems on lattices that are used to create public key cryptographic schemes from lattices. The most commonly used of these problems is the Shortest Vector Problem.

Definition 3.5. SVP
Let $\mathcal{L}$ *be a lattice and* B *an arbitrary basis of* $\mathcal{L}$*.*
Problem: Given B*, find* $\vec{v}$ *with* $\|\vec{v}\| = \lambda_1(\mathcal{L})$

To solve the problem the shortest non-zero vector in the lattice has to be found. In Figure 3.1 this would be $\vec{a_2}$. However, if the given base is not of such a good shape, the problem becomes much harder, this also applies to higher dimensional lattices. In addition to the SVP there is the more general Closest Vector Problem, which is illustrated in Figure 3.2.

Definition 3.6. Closest Vector Problem (CVP)
Let $\mathcal{L}$ *be a lattice with dimension* n *and* B *an arbitrary basis of* $\mathcal{L}$*.*
Problem: Given a vector $\vec{t} \in \mathbb{R}^n$ *find the vector* $\vec{v}$ *in the lattice* $\mathcal{L}(B)$ *that is closest to* $\vec{t}$*.*

How hard it is to solve the CVP again depends on the basis B, as with the SVP, the shorter and more orthogonal the basis vectors are, the more efficiently the problem can be solved [28]. Thus the relevant question is how to calculate a «good» base B' from any base B. So far the best known algorithm for this lattice reduction is the 1982 published LLL algorithm from Lenstra, Lenstra and Lovász [23]. It runs in poly time and gives an sub-exponential approximation by the factor $\gamma(n) = 2^{\mathcal{O}(n)}$, where n is the dimension of the lattice. This is acceptable since for lattice-based cryptography requires only a polynomial approximation problem factor $\gamma(n) \geq n$ to be secure [33]. The best known algorithms which provide an approximation within a polynomial factors for n have a run time of $2^{\Theta(n \log n)}$ or $2^{\Theta(n)}$ and also need exponentially space [1], which makes them more or less impractical to use, with higher dimensions. Now the following section will explain how to use these computational hard problems to build public key cryptography with lattices.

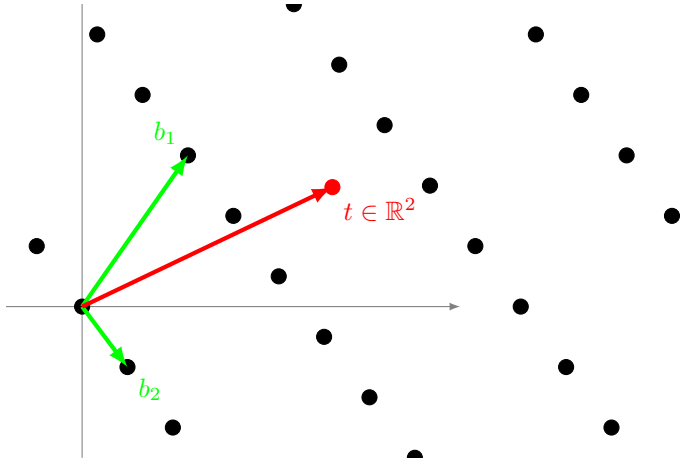


Figure 3.2: The Closest Vector Problem in a two dimensional lattice, given $\vec{t}$ find the closest point in $\mathcal{L}(B)$ with $B = (\vec{b_1}, \vec{b_2})$.

3.2 Ring-Learning with Errors(RLWE)

To build a public key scheme based on the RLWE technique either cyclic or ideal lattices are used. These are lattices are created by using an ideal Ring $\mathcal{R}$ of the form $\mathbb{Z}[x]/\mathbb{Z}p$, with p being a irreducible polynomial of degree n [24], which then can be mapped to a lattice with an additive isomorphism [25]. The cyclic rings $\mathcal{R}_q$ used for RLWE schemes are even more specific. They will only consist out of up to $n-1$ coefficients with the values in $[0, q-1]$, where q is a sufficiently large prime number and n a power of 2. The RLWE idea was introduced by Lyubashevsky, Peikert and Regev in 2010 to improve the already existing learning with errors approach [25], because the existing learning-with-errors approach has the same hardness as the worst case lattice problems, but its quadratic communication overhead reduces its practicality. The basic concept is the same, a small secret error is applied to an element making it hard to distinguish from a uniform randomly selected element. However, by knowing the secret, information can still be learned from the «erroneous» element.

So if Alice and Bob want to exchange information, we assume both know a public $\mathbf{a} \in \mathcal{R}_q$. Then first, Alice samples a secret $\mathbf{s}$ and an error $\mathbf{e}$ and then Bob as well samples his $\mathbf{s}'$ and $\mathbf{e}'$ from a distribution χ over $\mathcal{R}_q$. Next they create their public keys $\mathbf{b} = \mathbf{as} + \mathbf{e}$ respectively $\mathbf{u} = \mathbf{as}' + \mathbf{e}'$ and send them to each other. Now both can compute nearly the same value $\mathbf{v}' = \mathbf{us}$ and $\mathbf{v} = \mathbf{bs}'$. By using a Gaussian distribution for $\mathcal{X}$, the secrets $\mathbf{s}, \mathbf{s}'$ and errors $\mathbf{e}, \mathbf{e}'$ will be relatively small compared to $\mathbf{a}$. This way $\mathbf{v}'$ is approximately the same as $\mathbf{v}$. The whole protocol is illustrated in Figure 3.3.

- Let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$
- Let $\mathcal{X}$ be a distribution on $\mathcal{R}_q$ with small coefficients

Alice		**Bob**
	chose public $\mathbf{a} \in \mathcal{R}_q$	
$\mathbf{s}, \mathbf{e} \xleftarrow{R} \mathcal{X}$		$\mathbf{s}', \mathbf{e}' \xleftarrow{R} \mathcal{X}$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{\mathbf{b}}$	$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{\mathbf{u}}$	$\mathbf{v} \leftarrow \mathbf{bs}'$

$$\mathbf{v}' = \mathbf{us} = \mathbf{as}'\mathbf{s} + \mathbf{e}'\mathbf{s} \approx \mathbf{ass}' + \mathbf{es}' = \mathbf{bs}' = \mathbf{v}$$

Figure 3.3: The basic principle of a Ring-Learning with Errors scheme

The security of the RLWE is based on the Ring-Learning with Errors Decision problem, which can be reduced to the SVP [32, Theorem 2.7].

Definition 3.7. Ring-LWE Distribution [32]
For an $\mathbf{s} \in \mathcal{R}_q$ and a distribution $\mathcal{X}$ over $\mathcal{R}_q$, a sample from the RLWE distribution $A_{s,\chi}$ over $\mathcal{R}_q \times \mathcal{R}_q$ is generated by choosing $\mathbf{a} \xleftarrow{R} \mathcal{R}_q$, choosing $\mathbf{s}, \mathbf{e} \xleftarrow{R} \mathcal{X}$, and outputting $(\mathbf{a}, \mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e})$

Definition 3.8. Ring-LWE, Decision problem [32]
The decision version of the RLWE problem, denoted R-DLWE$_{q,\chi}$, is to distinguish with non-negligible advantage between independent samples from $A_{s,\chi}$, where $\mathbf{s} \xleftarrow{R} \mathcal{X}$ is chosen once and for all, and the same number of uniformly random and independent samples from $\mathcal{R}_q \times \mathcal{R}_q$.

3.3 NewHope-simple

This section explains a concrete implementation of an RLWE scheme, which is NewHope-simple. It is one of the 26 submissions of the second round of the NIST-competition mentioned in section 2.3 and it can be used as a KEM. For NewHope a polynomial ring $\mathcal{R}_q$ with the parameters (N, q) is used. NewHope$_{1024}$ is using $(1024, 12289)$ and NewHope$_{512}$ $(512, 12289)$ so both use the same prime but different degree of the polynomial. To sample the errors and the secret keys a centered binomial distribution ψ_8^N is used, which behaves nearly like a Gaussian distribution. The reason why no Gaussian distribution is used is that it cannot be implemented efficiently and safely [3]. As a higher dimension improves the security for the rest of the work NewHope$_{1024}$ is used and denoted as NewHope. The description is based on the NewHope-simple paper [2] and the project report [40].

The following only describes the CPA-secure scheme, as the CCA-security is achieved by a variant of the Fujisaki-Okamoto transformation [16] made by Hofheinz et al. [20], which uses a CPA-secure scheme to create a CCA-secure one. In order to describe the entire procedure, a few algorithms must first be introduced.

Key Encode and Decode *Key Encode* encodes a bit string ν of length $n = \frac{N}{4} = \frac{1024}{4} = 256$ into a polynomial $\mathbf{k} \in \mathcal{R}_q$. To make it possible for *Key Decode* to recover ν from an noisy element $\mathbf{k}'$, ν will be encoded 4 times into $\mathbf{k}$, by setting $k[i] = k[i+n] = k[i+2n] = k[i+3n]$ to either $\frac{q}{2}$ if $\nu[i] = 1$ or to zero otherwise. *Key Decode* will sum the absolute value of the four relevant indices up and decide if it is closer to zero or $2q$. Algorithm 3.1 and 3.2 show the detailed procedure of *Key Encode/Decode*.

Algorithm 3.1 Key Encode

Input: $\nu \in \{0,1\}^n : n = 256$

$k \leftarrow 0$
for $i := 0$ to $n-1$ **do**
  $\mathbf{k}[i] \leftarrow \nu[i]\lfloor \frac{q}{2} \rfloor$
  $\mathbf{k}[i+n] \leftarrow \nu[i]\lfloor \frac{q}{2} \rfloor$
  $\mathbf{k}[i+2n] \leftarrow \nu[i]\lfloor \frac{q}{2} \rfloor$
  $\mathbf{k}[i+3n] \leftarrow \nu[i]\lfloor \frac{q}{2} \rfloor$
end for

Return: k

Algorithm 3.2 Key Decode

Input: $\mathbf{k} \in \mathcal{R}_q$

$\nu \leftarrow 0$
for $i := 0$ to $n-1$ **do**
  $t \leftarrow \sum_{j=0}^{3} |\mathbf{k}[i+256j] - \lfloor \frac{q}{2} \rfloor|$
  if $t < q$ **then** $\nu_i \leftarrow 1$
  else $\nu_i \leftarrow 0$
  end if
end for

Return: ν

Compress and Decompress To reduce the amount of exchanged data, some elements get compressed before and decompressed after the transmission. As the relevant information is stored in the most significant bits, the compression is achieved by performing a modulus switching from a polynomial $\mathbf{k} \in \mathcal{R}_q$ to an element $\mathbf{c} \in \mathbb{Z}_8[x]/(x^N+1)$. So only the three most significant bits are kept. The decompress algorithm then shifts the 3 bits back into the full modulus of $\mathcal{R}_q$. The detailed procedure can be seen in algorithm 3.3 and algorithm 3.4.

Algorithm 3.3 Compress

Input: $\mathbf{m} \in \mathcal{R}_q$

for $i := 0$ to $N-1$ **do**
  $\mathbf{c}[i] \leftarrow \lceil \frac{8 \cdot \mathbf{m}[i]}{q} \rfloor \bmod 8$
end for

Return: $\mathbf{c} \in \mathbb{Z}_8[x]/(x^N+1)$

Algorithm 3.4 Decompress

Input: $\mathbf{c} \in \mathbb{Z}_8[x]/(x^N+1)$

for $i := 0$ to $N-1$ **do**
  $\mathbf{m}'[i] \leftarrow \lceil \frac{q \cdot \mathbf{c}[i]}{8} \rfloor$
end for

Return: $\mathbf{m}' \in \mathcal{R}_q$

Like all KEM schemes, NewHope has 3 poly-time algorithms, *Key Generation*, *Encapsulation*, *Decapsulation*, whose execution can be described in three steps.

1. *Key Generation:*
   The public parameter $\mathbf{a} \xleftarrow{R} \mathcal{R}_q$ gets chosen by Alice. In the practical realization, this is done by a *Parse* function, which takes a random seed as input, hashes it and interprets the hash value as coefficients of a polynomial in $\mathcal{R}_q$. Additionally she samples her secret key $\mathbf{s} \xleftarrow{R} \psi_8^N$ and a small error $\mathbf{e} \xleftarrow{R} \psi_8^N$ to calculate her public key $\mathbf{b} = \mathbf{as} + \mathbf{e}$. The pair $(\mathbf{a}, \mathbf{b})$ then is sent to Bob.

2. *Encapsulation:*
   Bob picks a $\nu_B \xleftarrow{R} \{0,1\}^n$ which will be the basis for the later shared key and samples $\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{R} \psi_8^N$. Next he computes his public key $\mathbf{u} = \mathbf{as}' + \mathbf{e}'$ and encodes ν_B into $\mathbf{k}$ with the *Key Encode* algorithm. He then computes the ciphertext $\mathbf{c} = \mathbf{bs}' + \mathbf{e}'' + \mathbf{k}$, which gets compressed into $\hat{\mathbf{c}}$ with the *Compress* algorithm. The pair $(\mathbf{u}, \hat{\mathbf{c}})$ then is sent to Alice.

3. *Decapsulation*:
   Alice decompresses $\hat{\mathbf{c}}$ to $\mathbf{c}'$ with the *Decompress* algorithm to then calculate $\mathbf{k}' = \mathbf{c} - \mathbf{us}$ such that:

   $$\mathbf{k}' = \mathbf{c} - \mathbf{us} = (\mathbf{as} + \mathbf{e})\mathbf{s}' + \mathbf{e}'' + \mathbf{k} - (\mathbf{as}' + \mathbf{e}')\mathbf{s} = \mathbf{k} + \mathbf{es}' + \mathbf{e}'' - \mathbf{e}'\mathbf{s} \quad (3.1)$$

   Then she extracts ν_A from $\mathbf{k}'$ with the *Key Decode* algorithm, which is possible as the coefficients in $\mathbf{es}' + \mathbf{e}'' - \mathbf{e}'\mathbf{s}$ are rather small compared to $\mathbf{k}$. ν_A is then used to derive the shared secret key μ_A by hashing it with SHA3-256 [8]. The hashing is done to prevent a leakage of Bob random number generator state, as the final shared secret key only relies on Bob's random input.

In Figure 3.4 the full protocol description is provided. However, one simplification has been made. The full implementation of the protocol uses the Number Theoretic Transform (NTT) [17] in order to perform the multiplication of polynomials more efficiently. Since the transformation to the Number Theoretic Transform (NTT) domain has no influence on the security properties of NewHope, it is omitted here.

3.4 Original NewHope

The above described version of NewHope is the one that was submitted to the second round of the NIST competition. It is called NewHope-simple because the authors think the new version is simpler that the original one from the first round. Although the main focus of this work is on NewHope-simple, the original variant is briefly introduced here. Because section 4.3 presents signal leak attack, which only works with the original variant.

Both versions rely on the same lattice construction with the same parameters. The difference is how the shared secret key is derived. While in NewHope-simple the

Parameters: $q = 12289, N = 1024, n = \frac{1024}{4} = 256$ with ψ_8^N

Alice		**Bob**
1. *Key Generation*		
$seed \xleftarrow{R} [0,255]^{32}$		
$\mathbf{a} \leftarrow Parse(seed)$		
$\mathbf{s}, \mathbf{e} \xleftarrow{R} \psi_8^N$		2. *Encapsulation*
		$\nu'_B \xleftarrow{R} \{0,1\}^n$
$\mathbf{b} = \mathbf{as} + \mathbf{e}$	$\xrightarrow{m_A = (\mathbf{b}, seed)}$	$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{R} \psi_8^N$
		$\mathbf{a} \leftarrow Parse(seed)$
		$\mathbf{u} = \mathbf{as}' + \mathbf{e}'$
		$\nu_B \leftarrow \text{SHA3-256}(\nu'_B)$
		$\mathbf{k} \leftarrow Key\ Encode\ (\nu_B)$
		$\mathbf{c} = \mathbf{bs}' + \mathbf{e}'' + \mathbf{k}$
3. *Decapsulation*		
$\mathbf{c}' \leftarrow Decompress(\hat{\mathbf{c}})$	$\xleftarrow{m_B = (\mathbf{u}, \hat{\mathbf{c}})}$	$\hat{\mathbf{c}} \leftarrow Compress(\mathbf{c})$
$k' = \mathbf{c}' - \mathbf{us}$		
$\nu_A \leftarrow Key\ Decode(\mathbf{k}')$		
$\mu_A \leftarrow \text{SHA3-256}(32, \nu_A)$		$\mu_B \leftarrow \text{SHA3-256}(32, \nu_B)$

Figure 3.4: NewHope-simple

ν_B gets chosen by Bob, who encodes and encrypts it into $\mathbf{c}$, the original version used a technique called reconciliation, where Bob generates a signal vector ω instead of a ciphertext $\hat{\mathbf{c}}$. To do so the signal function $Sig(\mathbf{v})$ is used.

Definition 3.9. Signal function of original NewNope
Let q be the prime of the ring $\mathcal{R}_q$ and $\mathbf{v} \in \mathcal{R}_q$.

$$Sig(v)[i] = \begin{cases} 0 & \text{if } \mathbf{v}[i] \in \{-\frac{q}{4}, \ldots, \frac{q}{4}\} \\ 1 & \text{otherwise} \end{cases}$$

Bob calculates $\mathbf{k_B} = \mathbf{bs}' + 2\mathbf{e}''$ and $\omega = Sig(\mathbf{k_B})$ to send $m_B = (\mathbf{u}, \omega_B)$.
Then Alice also calculates $\mathbf{k_A} = \mathbf{us}$ such that both can use $k_{A/B}$, ω and the reconciliation function of the form Mod_2: $(\mathbb{Z}_{\text{ıı}} \times \{0,1\}) \to \{0,1\}$ to derive the shared secret key μ. The full procedure is presented in Figure 3.5.

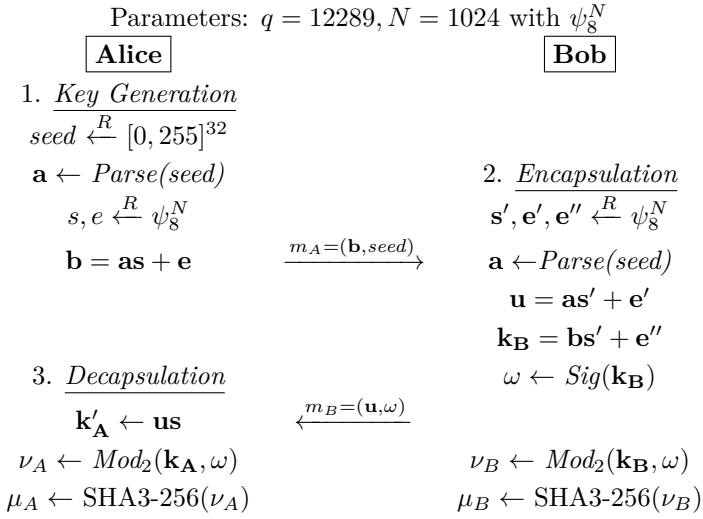


Figure 3.5: original version of NewHope

Chapter 4
Basic attacks

This chapter starts with a general introduction to decoding failures and their effects on the security of cryptographic schemes. This is followed by a description of an attack idea against the original version of NewHope. The rest is devoted to the attack against NewHope simple published by Bauer et al. The attack is not only presented but also the problems that occur during the reproduction are explained. Furthermore, the results of the restoration are also presented.

4.1 Decoding failures

In general as soon as a cryptographic scheme uses a decoding algorithm within it's encryption or decryption algorithm there is a possibility of decoding failures. This could can lead to problems as only legitimate identities involved in the protocol should be able to encode a ciphertext, which can later be decoded correctly by the respective party. Especially those failures are relevant where the decoding algorithm delivers a supposedly valid output instead of outputting $\perp$ together with an error message.

There are two options to address the concerns about this problem. First it can be shown that the occurrence of decoding failures is not possible within the scope of the protocol. Second it can be stated that it only occurs with a certain probability that can be considered as acceptable. Since almost all schemes use some random input to create keys or ciphertexts, the first option is often not possible or at least quite complex. Therefore the probability-based variant is usually chosen. Then it is often stated that one out of a given number of decoding attempts fails. Here it also plays a role whether decoding failures can be detected and possibly corrected with other methods or not, but usually the protocol is simply executed again.

On the other hand, decoding failures can also reveal information about internal states, which could be interesting from the security perspective. It generally cannot be said whether the extractable information can be used for an attack or not, as

it depends strongly on the particular scheme. However, the past has shown that decoding failures can be used for successful attacks. post-quantum schemes are no exception. An example is the attack by Guo et al. [18] against the QC-MPCD variant of McEliece, which is one of the oldest PQ schemes based on linear codes, where they first created special patterns to query the decapsulation oracle of a weakened variant of the scheme. Based on the eventually occurring decoding failures, it was possible to reconstruct the secret key. Next they could transfer this attack to the CCA variant.

In the specific case of NewHope decoding failures can only be detected after the end of the protocol, as each party has only its $\mu_{A/B}$ and cannot determine whether $\mu_A = \mu_B$. However, μ_A and μ_B are mostly intended as key for the symmetric encryption of the following communication, so it should be possible for them to determine whether the other communication partner has the same key or not. The usability of NewHope depends on how often a decoding failure occurs. Therefore the authors of NewHope have done an analysis and concluded that the probability of a decoding failure is 2^{-60}. Considering that NewHope is intended for the use on the internet, is an acceptable failure probability, since the authors of NewHope said that other failures are much more likely to occur on the internet [3].

4.2 Key reuse

A common pattern for creating attacks on cryptographic procedures can be observed here. First, a successful attack on a weakened scheme is developed in order to be use as a sub step for an attack on the full scheme in the second step. A similar approach is used for the later described attacks on New Hope. Also, for lattice-based cryptographic schemes decoding failure attacks have been found. For example, there is one on the NTRU scheme, where the secret key can be derived from the number of decodings [21].

All these attacks are based on one common assumption: That the secret key will be reused by the honest party on a long-term basis, making it possible to do many decoding queries. For a public key scheme this assumption is quite realistic, because usually the goal of to create the public/private key pair once and then use it for a longer period of time. Although it is possible to generate new keys after each decryption, the new public key would have to be sent to the communication partners again. This massively increases the communication overhead with corresponding limitations in practicability.

However, this works differently for KEMs. These, are usually used at the beginning of the communication and most of the protocols foresee sending the public keys anyway, making it acceptable to generate new keys for each protocol execution. Nevertheless, the generation of new keys still requires computing time. Although this

may not be particularly problematic for a single communication, it may be worth using the generated key pair over a longer period of time, if it is considered that a server may have to establish thousands of connections in a short period of time. For this reason, a draft for the TLS 1.3 protocol includes the feature to use keys over a longer period of time [37]. Even if this is only an optional feature, it can be assumed that the keys will be used for a longer period on the server side. In addition, the integration of cryptographic schemes into server software is often quite complex and the key management may not be taken over by the eventually secure and correct implementation of the scheme. Therefore, also for KEMs the assumption can be made that the keys will be reused. This class of attacks is then called key-reuse attacks.

Since NewHope contains an encoding and decoding algorithm, and is intended to be used on servers with the TLS protocol it is reasonable to analyze if decoding failures can leak security-related information under the assumption of key reuse. In the following section different approaches for such a attack are presented.

4.3 Signal Leakage

The signal leakage attack that was published by Fluhrer [15] and extended by Ding et al. [13], is against RLWE schemes that use a signal function like the original version of NewHope described in section 3.4 and outlined in Figure 3.5. The signal vector ω is used to learn information about the secret key s and eventually fully recover it. While the attack proposed by Fluhrer is attacking Alice to recover her secret key, the version of Ding et al. is targeting Bob. Since the variant of Ding is the more advanced one, this section is limited to his variant and gives a simplified outline of the attack in the following.

As the attack is a key-reuse attack it assumes that Bob uses his key for a longer period of time, so the attacker $\mathcal{A}$ is in the position of Alice and initiates several key exchanges to learn information about Bob's secret key. All the parameters are the ones given from the NewHope description above. The procedure can be divided into four steps, assuming that the public parameter $\mathbf{a}$ is already known by the attacker $\mathcal{A}$ and Bob.

Step 1: $\mathcal{A}$ sets his $\mathbf{s} = 0$ and $\mathbf{e} = 1 \in \mathcal{R}_q$ as the neutral element in the ring. Additionally he defines a $\mathbf{k} \in \mathcal{R}_q$ to create his public key $\mathbf{b} = \mathbf{as} + \mathbf{ke} = \mathbf{k}$.
Which will result in Bob's $\mathbf{k_b}$ being $\mathbf{k_b} = \mathbf{bs'} + \mathbf{e''} = \mathbf{ks'} + 2\mathbf{e''}$.
This configuration is then used to initiate $q - 1$ key exchanges, where the individual coefficients $\mathbf{k}[i]$ are iterated over form 0 to $q - 1$.
Since the signal vector is calculated from k_B, the signal for an element changes whenever $|k\mathbf{s'}[i]|$ is close to $\frac{q}{4}$, according to the signal function from Definition

3.4. So by counting the signal changes in signal vector ω $\mathcal{A}$ can now calculate the individual absolute values $\mathbf{s}'[i]$. Thus, per coefficient $\mathbf{s}'[i]$ signal changes occur. However, if $|\mathbf{ks}'[i]|$ is in the range around $\frac{q}{4}$ there may be more frequent signal changes caused by the error $\mathbf{e}''$, but according to Ding et al. it is possible to just ignore these areas with high frequent signal changes [13].

Step 2: After determining the absolute values of $\mathbf{s}'$, it is now necessary to find the signs of the coefficients. To do so, step one gets repeated with one configuration change, the public key will be $\mathbf{b}' = (1 + x)\mathbf{b}$. This will give $\mathcal{A}$ the signal changes cause by $(1 + x)\mathbf{ks}'$. Making it possible for him to compute the absolute value of the coefficients of $(1+x)\mathbf{s}' = \{\mathbf{s}'[0]-\mathbf{s}'[N-1], \mathbf{s}'[1]+\mathbf{s}'[2], \ldots, \mathbf{s}'[N-2]+\mathbf{s}'[N-1]\}$.

Step 3: The information gained in the first two steps now can be combined. $\mathcal{A}$ knows $|\mathbf{s}'[0]|, |\mathbf{s}'[N-1]|$ and $|\mathbf{s}'[0] - \mathbf{s}'[N-1]|$ so he can determine whether $\mathbf{s}'[0]$ and $\mathbf{s}'[N-1]$ have the same or a different sign.
For all other pairs $\mathbf{s}'[x], \mathbf{s}'[y] : x \in [0, N-2], y \in [1, N-1]$, it can be determined with the help of the information $|\mathbf{s}'[x] + \mathbf{s}'[y]|$ from step 2 whether they have the same or different signs.

Step 4: Finally $\mathcal{A}$ knows all the signs of all $\mathbf{s}'[i]$ relative to its neighbours, which results in two possibilities for $\mathbf{s}'$. By testing $\mathbf{s}[0]$ and $-\mathbf{s}[0]$ $\mathcal{A}$ can now find out which one is the correct one and $\mathbf{s}'$ is completely recovered.

4.4 Attacks on NewHope-simple

Since NewHope encodes and encrypts the shared key into a ciphertext and does not use a signal vector, another source of information must be used. As mentioned before, decoding failures in NewHope cannot be detected during the protocol, but during the following communication it can be verified if the keys of Alice and Bob match or not. With this one bit of information a key mismatch attack can be created as shown by Bauer et al. [4]. They focus on an attack where Alice, who reuses her secret keys, gets attacked by a malicious Bob. In reality Alice would be the server and Bob the client. Nevertheless, there is also an attack for the scenario that Bob reuses his key and communicates with a malicious Alice even though this case is less likely to happen. Both attacks target the CPA secure variant of NewHope and not the final CCA secure scheme, but they are still worth considering, since attacks on a weakened scheme often lead to advancements for an attack against the more secure variant of the scheme. As the attack against Bob is less complicated than the one against Alice it will be explained first and is followed by description of the more sophisticated attack against Alice.

4.4.1 Malicious Alice

The basic idea for this key-reuse attack was published by Ding et al. in 2017 [13] and adapted for NewHope by Bauer et al. [4]. It is assumed that Bob uses his secret key $\mathbf{s}'$ and the error $\mathbf{e}'$ for multiple connections to different servers. In contrast to the Signal Leakage attack previously presented in section 4.3, this key reuse attack does not involve an active attacker A, but only a passive eavesdropper Eve. It reads the messages between Bob and two different servers, which allows her to calculate the private key $\mathbf{s}'$ as follows.

1. Eve collects the message from the first server $m_{1,A_1} = (\mathbf{b_1}, \mathbf{a_1}) = Parse(\text{SHA3-256}(seed_1))$ and the reply $m_{1,B} = (\mathbf{u_1}, \mathbf{c_1})$ from Bob

2. Then she collects $m_{2,A_2} = (\mathbf{b_2}, \mathbf{a_2}) = Parse(\text{SHA3-256}(seed_2))$ with $\mathbf{a_1} \neq \mathbf{a_2}$ from the second server and the corresponding $m_{2,B} = (\mathbf{u_2}, \mathbf{c_2})$ from Bob.

3. As $\mathbf{e}''$ is sampled from ψ_8^N, it is considerably small and gets removed by *Compress* , thus it needs not be considered.
   Then Eve can calculate $\mathbf{s}'$ directly:

$$\mathbf{s}' = \frac{\mathbf{u_1} - \mathbf{u_2}}{\mathbf{a_1} - \mathbf{a_2}} = \frac{(\mathbf{a_1 s}' + \mathbf{e}') - (\mathbf{a_2 s}' + \mathbf{e}')}{\mathbf{a_1} - \mathbf{a_2}}$$

Although the attack does not necessarily require an active attacker $\mathcal{A}$, it can be performed by one. The attacker has to execute the protocol twice with Bob and has to make sure that he uses two different $\mathbf{a_1} \neq \mathbf{a_2}$. Then he can calculate the secret key as in described step 3.

4.4.2 Malicious Bob

The attack of Bauer et al. against an honest Alice is more complex and requires an active attacker $\mathcal{A}$ that is in Bob's position. The corresponding scenario is that a server (in our case Alice) uses her secret key for all her connections over a longer period of time. If the attacker succeeds in recovering the secret key of Alice, he is able to decrypt the messages from the remaining connections of Alice. The general approach is that $\mathcal{A}$ guesses a key value $\nu_{\mathcal{A}'}$ and creates a special corresponding public key $\mathbf{u}$ and ciphertext $\hat{\mathbf{c}}$, which he sends to Alice. Depending on whether the key created by Alice matches or not, Bob repeats this process and can gradually extract parts of Alice's secret key $\mathbf{s}$. This gave the attack its name «key mismatch attack», as it is based on the technique that the attacker checks whether he gets a key mismatch or not. The fully detailed description that follows is based on the work of Bauer et al. [4].

Definition 4.1. Key mismatch oracle
A key mismatch oracle is an oracle that outputs one bit of information on the possible mismatch at the end of the key encapsulation mechanism.

In a practical implementation of the attack, the attacker $\mathcal{A}$ would determine whether the key he guessed and the key derived from Alice matches or not by checking if he can decrypt the messages of the following communication from Alice to something meaningful or not. Assuming that $\mathcal{A}$ can get this information, it is possible to combine the key mismatch step and the decapsulation part of Alice into a key mismatch oracle. This simplifies the theoretical description of the attack by giving the attacker the public parameter **a** and Alice public key **b** as input and instead of interacting with a «real» Alice he can use the key mismatch oracle to get information about a possible key mismatch. The full procedure of the attack model is given in Figure 4.1.

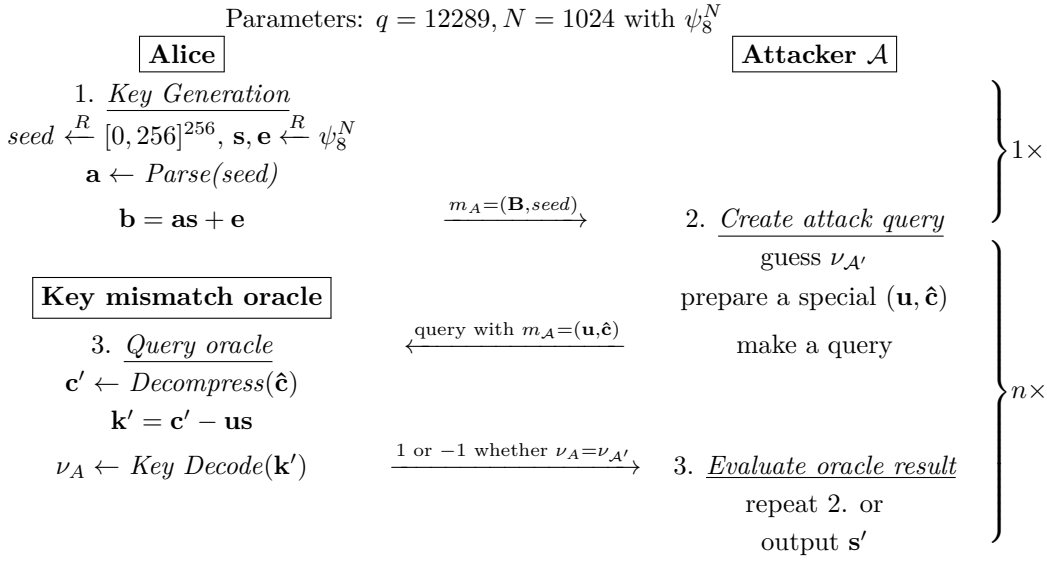


Figure 4.1: The procedure of the attack on Alice secret key **s**

Creation of the key mismatch oracle

The key mismatch oracle will be defined in several stages, starting with $\mathcal{O}_1$.

Definition 4.2. Key mismatch oracle $\mathcal{O}_1$

$$\mathcal{O}_1(m_{\mathcal{A}}, \mu_{\mathcal{A}}) = \begin{cases} 1 \text{ if } Decapsulation(m_{\mathcal{A}'}, \mathbf{s}) = (\nu_A) = \nu_{\mathcal{A}'} \\ -1 \ otherwise \end{cases}$$

As already mentioned, $\mathcal{A}$ makes a fixed guess for the value of $\nu_{\mathcal{A}'}$ that is why from now on the value of $\nu_{\mathcal{A}'}$ is fixed according to Equation 4.1

$$\nu_{\mathcal{A}'} = (1, 0, \ldots, 0) : |\nu_{\mathcal{A}'}| = 256 \tag{4.1}$$

Why he chooses this particular string is explained later together with the verification of Hypothesis 4.4. Together with the definition of $m_{\mathcal{A}} = (\mathbf{u}, \hat{\mathbf{c}})$ the next variant of the oracle can be defined.

Definition 4.3. Key mismatch oracle $\mathcal{O}_2$

$$\mathcal{O}_2(\mathbf{u}, \hat{\mathbf{c}}) = \mathcal{O}_1((\mathbf{u}, \hat{\mathbf{c}}), \nu_{\mathcal{A}'})$$

$\mathcal{A}$ has his public key $\mathbf{u}$ and the ciphertext $\hat{\mathbf{c}}$ to influence the output of $\mathcal{O}_2$. To understand which possible combinations of $\mathbf{u}$ and $\hat{\mathbf{c}}$ can help to obtain information about the secret key $\mathbf{s}$, a closer look is taken into the process of how Alice computes her key value ν_A, especially how the individual bits $\nu_A[i]$ are deducted.

$$\nu_A[i] = Sign\left(\sum_{j=0}^{3} \left|(Decompress(\hat{\mathbf{c}}) - \mathbf{us})\left[i + nj\right] - \left\lfloor \frac{q}{2} \right\rfloor \right| - q\right) \tag{4.2}$$

The $Sign$ function outputs an 0 for for all all numbers ≥ 0 and a 1 otherwise. Later this will be mapped just to a $+$ instead of a 0 and $-$ instead of a 1. Since the attacker only gets one bit of information from the oracle, but there are potentially $n = 256$ bits that can cause a key mismatch, he will need to ensure that he knows $n - 1$ bits of ν_A, to have only one bit of variation. To achieve this, a hypothesis about Alice's ν_A is formulated.

Hypothesis 4.4.

$$\nu_A[i] = 0 : i \in [1, n-1]$$

Additionally it has to be considered that the *Key Encode* and *Key Decode* algorithm combine four coefficients of $\mathbf{s}$ into one bit of ν_A. This means the smallest number of coefficients of $\mathbf{s}$ that can be targeted at once is four. To do so the quadruplet $l = (l_0, l_1, l_2, l_3)$ for a target index k corresponding to $\mathbf{s}[k + nj]_{j=0\ldots3}$, is used by the attacker to choose his public key $\mathbf{u}$ and a $\hat{\mathbf{c}}$ such that hypothesis 4.4 is verified. This can be achieved by setting them as stated in Equation 4.3.

$$\mathbf{u} = \left\lfloor \frac{q-1}{8} \right\rceil x^{-k} \text{ and } \hat{\mathbf{c}} = \sum_{j=0}^{3} ((l_j + 4) \bmod 8) \cdot x^{nj} \tag{4.3}$$

As $\mathbf{u}$ gets multiplied with $\mathbf{s}$, the x^{-k} is used to shift the $\mathbf{s}[k + nj]_{j=0\ldots3}$ to $\mathbf{s}[nj]_{j=0\ldots3}$. These are exactly the coefficients used to decode $V_A[0]$. For this reason

only the coefficients $\hat{\mathbf{c}}[nj]_{j=0\ldots3}$ are of interest for the ciphertext. All other coefficients in $\hat{\mathbf{c}}$ will be set to zero, such that the likelihood that Alice decodes them to zero is high. Bauer et al. discussed how likely this is and came to the conclusion that in at least 94.6% of the cases $\mathbf{s}$ will be in a form that Hypothesis 4.4 will be fulfilled [4]. With this the final key mismatch oracle $\mathcal{O}$ can be formulated as a variant of $\mathcal{O}_2$ depending on the target index k and the quadruplet l.

$$\mathcal{O}(k,l) = \mathcal{O}_2\left(\left\lfloor\frac{q-1}{8}\right\rceil x^{-k}, \quad \sum_{j=0}^{3}((l_j+4) \bmod 8)\cdot x^n j\right)$$

$$\mathcal{O}(k,l) = Sign\left(\sum_{j=0}^{3}\left|(Decompress(\hat{\mathbf{c}}) - \mathbf{us})[k+nj] - \left\lfloor\frac{q}{2}\right\rceil\right| - q\right)$$

Running the algorithm on the values from Equation 4.3 into the *Decompress* algorithm

$$Decompress(\hat{\mathbf{c}})[nj] = \left\lfloor\frac{(l_j+4 \mod 8)\cdot q}{8}\right\rceil = (l_j+4 \mod 8)\cdot\frac{q-1}{8}$$

and plugging the values of $\mathbf{u}$ in:

$$\mathcal{O}(k,l) = Sign\left(\sum_{j=0}^{3}\left|\left(l_j+4) \mod 8\right)\frac{q-1}{8} - \frac{q-1}{8}\mathbf{s}[k+nj]\right) - \left\lfloor\frac{q}{2}\right\rceil\right| - q\right)$$

With q divided by $\frac{q-1}{8}$ being ≈ 8.0007 and $\left\lfloor\frac{q}{2}\right\rceil = \frac{q-1}{2}$, the whole formula can be divided by $\frac{q-1}{8}$.

$$\mathcal{O}(k,l) = Sign\left(\sum_{j=0}^{3}|((l_j+4) \bmod 8) - \mathbf{s}[k+nj]) - 4| - 8\right)$$

If l gets limited to $[-4,3]^4$ then $(l_j+4) \bmod 8 = (l_j+4)$ so:

$$\mathcal{O}(k,l) = Sign\left(\sum_{j=0}^{3}|l_j - \mathbf{s}[k+nj]| - 8\right)$$

Definition 4.5. Final key mismatch oracle
Assumed Hypothesis 4.4 is verified. Let $k \in [0, n-1]$ be the target index, $\hat{\mathbf{c}}$ set as

described in equation 4.3, let l be the integer quadruplet in $[-4,3]^4$. The final key mismatch oracle is:

$$\mathcal{O}(k,l) = Sign\left(\sum_{j=0}^{3} |l_j - \mathbf{s}[k+nj]| - 8\right)$$

By using the final key mismatch oracle $\mathcal{O}$ the attacker can create queries just depending on the quadruple l and the target index k iterating from 0 to 255.

Recovering the secret key with the help of the key mismatch oracle

This section describes how the attacker $\mathcal{A}$ can use the key mismatch oracle from Definition 4.5 to recover the secret key, by setting the values of the quadruple l from Equation 4.3 in a certain way. For simplicity's sake, the following assumes that k is fixed within $[0,255]$. Accordingly, the coefficients of $\mathbf{s}$ will be denoted as $s_j = \mathbf{s}[k+nj] : j \in [0,3]$. Although the coefficients of $\mathbf{s}$ are sampled from the distribution ψ_8 and thus lie in the interval $[-8,8]$, for the beginning we only consider the coefficients from the subset $[-3,2]$ and will expand the interval set later in section 4.4.2. Since the four coefficients are recovered one by one, we will only consider $_0$ as the the successive 3 coefficients are calculated analogously.

At best, it will take only eight queries to recover s_0. For the first query l_0 gets set to -4 and the other three elements will be sampled uniformly random from the interval $[-4,3]$. In the following seven queries l_0 will be increased by one until $l_0 = 3$. The according results from the oracle will look like one of the three following cases shown in Table 4.1.

l_0	-4	-3	-2	-1	0	1	2	3
i.)	+	+	+	+	+	+	+	+
ii.)	−	−	−	−	+	+	+	+
iii.)	+	+	+	−	−	−	+	+

Table 4.1: The three different results of a set of queries for one coefficient

To find out which of the three cases occurs, one has to recap how characters are calculated in $\mathcal{O}$.

$$\sum_{j=0}^{3} |l_j - \mathbf{s}[k+nj]| - 8$$

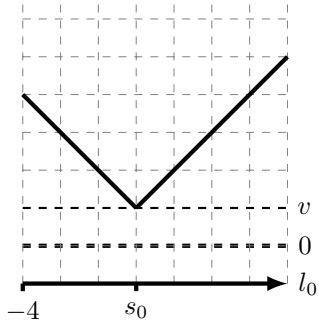
As l_1, l_2, l_3 are fixed over the eight queries, the last three elements of the sum can be expressed as a fixed $v = \sum_{j=1}^{3} |l_j - s_j| - 8$. This allows f_v to be defined as a function

of l_0. Which of the cases occurs depends on the values of $f_v(l_0)$

$$f_v(l_0) = |l_0 - s_0| + v \tag{4.4}$$

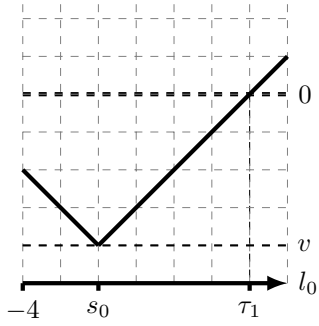
We can subdivide the dependency on v into three cases:

(i) v is greater than zero:
All queries have to result in a positive sign.



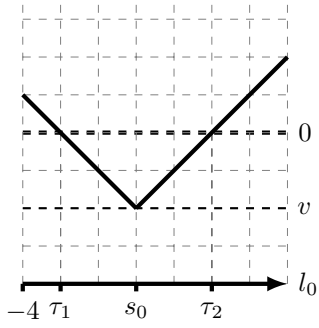
Values of f_v if $v \geq 0$

(ii) $v \ll 0$, v is a lot smaller then zero:
Then the first queries will be negative and the last ones might have a positive result, where a change of sign occurs on τ_1.



Values of f_v if $v \ll 0$

(iii) v is less then zero but still close to zero:
Then the first results will be positive and become negative at τ_1. At τ_2 they will get positive again, so $\tau_1 < \tau_2$.
This case is called a favorable case, because it allows to recover s_0.



Values of f_v if $v < 0$

If case (i) or (ii) occurs, $\mathcal{A}$ has to choose new random values for l_1, l_2 and l_3 and make eight new queries, to see if v has the correct value to get a favorable case.

Note that if s_0 is not in the interval $[-3, 2]$, which was assumed at the beginning of section 4.4.2, then a favorable case will never happen. So in a real implementation, $\mathcal{A}$ would try it several times and then move on to the next s_j if no favorable case occurs. In the event that a favorable case occurs, as shown in (iii), because of the symmetry of f_v, the coefficient s_o can be calculated using τ_1 and τ_2:

$$s_0 = \frac{\tau_1 + \tau_2}{2} \tag{4.5}$$

Bauer et al. have conducted a study on the probability of the success of a favorable case appearance and came to the conclusion that as long as all s_1, s_2, s_3 lie in $[-4, 4]$ a favorable case is always possible. Take for example if the values of l are

$$l_1 = s_1 - 2 \cdot Sign(s_1),\ l_2 = s_2 - 2 \cdot Sign(s_2),\ l_3 = s_3 - 3 \cdot Sign(s_3)$$

then v is -1, which results in a favorable case.

Example 4.6. Let the four relevant coefficients of (s_0, s_1, s_2, s_3) be $(-1, 0, 2, -1)$. If $(l_1, l_2, l_3) = (3, 1, -3)$ then $v = -2$, which will result in the following oracle output:

l_0	-4	-3	-2	-1	0	1	2	3
$f_v(l_0)$	3	2	1	0	1	2	3	4
$\mathcal{O}$	+	+	−	−	−	+	+	+

So $\tau_1 = -2$ and $\tau_2 = 0$, which means that:

$$s_0 = \frac{\tau_1 + \tau_2}{2} = \frac{-2 + 0}{2} = -1$$

Expanding the recoverable interval The problem with the method described so far is that the coefficients of $\mathbf{s}$ are drawn from ψ_8 and not from ψ_4, so they lie in the interval $[-8, 8]$ instead of $[-4, 4]$. This also affects the target coefficient s_0, since it is very likely that s_0 lies outside the assumed interval $[-3, 2]$.

For this reason, Bauer et al. [4] proposed a method to double the two intervals to allow to recover coefficients in $[-6, 4]$. This also enables the possibility to always get favorable case, as the interval for the allowed values for s_1, s_2, s_3 will also be doubled to $[-8, 8]$. Therefore the public key $\mathbf{u}$ from Equation 4.3 must be halved, so that the new $\mathbf{u}$ is:

$$\mathbf{u} = \left\lfloor \frac{q-1}{8} \right\rceil x^{-k} \cdot \frac{1}{2} = \left\lfloor \frac{q-1}{16} \right\rceil x^{-k} \tag{4.6}$$

This causes the computation of the key mismatch oracle to change to the following:

$$\mathcal{O} = Sign\left(\sum_{j=0}^{3} \left| l_j - \frac{\mathbf{s}[k+nj]}{2} \right| - 8 \right)$$

Which accordingly also influences the result of the oracle, since the minimum of the function f_v from Equation 4.4 is no longer limited to an integer. The number of negative results can be either even or odd, as demonstrated in Figure 4.2 and Figure 4.3. In order to calculate s_0, the attacker must first distinguished how many

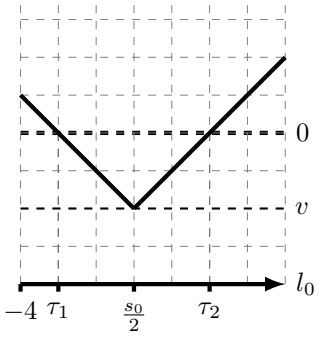


Figure 4.2: f_v with an even $\frac{s_0}{2}$

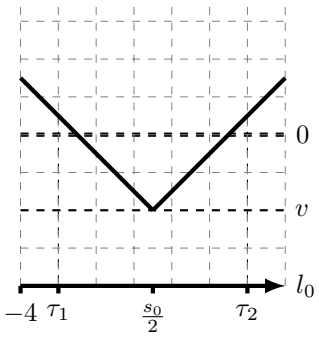


Figure 4.3: f_v with an odd $\frac{s_0}{2}$

times there is a negative result. To do so, he can simply check whether the sum $\tau_1 + \tau_2$ is even or odd.
If $\tau_1 + \tau_2 \mod 2 = 0$, then $\frac{S_0}{2}$ is in the set $\{-3, -2, -1, 0, 1, 2\}$ and the coefficient is calculated as follows:

$$s_0 = 2\frac{\tau_1 + \tau_2}{2} = \tau_1 + \tau_2$$

If $\tau_1 + \tau_2 \mod 2 = 1$, then $\frac{S_0}{2}$ is in the set $\{-2.5, -1.5, -0.5, 0.5, 1.5\}$ and the coefficient is calculated as follows:

$$s_0 = 2\left\lfloor \frac{\tau_1 + \tau_2}{2} \right\rfloor + 1$$

This analysis of the oracle outputs and calculation of the coefficient is formally described in the *FindS* Algorithm 4.1.

One disadvantage is that if s_0 is in $\{-8, 7, 5, 6, 7, 8\}$ $\mathcal{A}$ cannot recover s_0 with this method. However, as Table 4.2 shows, these values do not appear too frequently when sampling from a binomial distribution ψ_8 like the one used in NewHope. It shows that about 98% of the coefficients are within the recoverable range, which means concretely that ten of 1024 coefficients of $\mathbf{s}$ cannot be recovered. Those remaining coefficients must be found by a brute force search to find the complete key. As there are 6 possibilities for each of them the complexity for this search is $6^{10} \approx 2^{26}$, which is feasibly, but not optimal. The procedure from the attack side that recovers the recoverable coefficients is presented in algorithm 4.2.

value of s_0	0	1	2	3	4	5	6	7	8
Probability($\times 2^{16}$)	12870	11440	8008	4368	1820	560	120	16	1

Table 4.2: The probability distribution for the values 0 to 8 in a binomial distribution, created by Bauer et al. [4]

Algorithm 4.1 FindS

Input:oracle results b[8]

$\tau_1 \leftarrow \perp$
$\tau_2 \leftarrow \perp$
for $i := -3$ to 2 **do**
  if $b[i-1] = +$ and $b[i] = -$ **then** Finding the sign change from + to -
    $\tau_1 \leftarrow i$
  end if
  if $b[i] = -$ and $b[i+1] = +$ **then** Finding the sign change from - to +
    $\tau_2 \leftarrow i$
  end if
end for
$\tau \leftarrow \tau_1 + \tau_2$
if $\tau \mod 2 = 0$ **then**
  $c \leftarrow \tau$
end if
if $\tau \mod 2 = 1$ **then**
  $c \leftarrow 2\lfloor \frac{\tau}{2} \rfloor + 1$
else
  $c \leftarrow \perp$
end if

Return:coefficient c of s

Algorithm 4.2 Key recovery algorithm

```
for k := 0 to N − 1 do
    u ← (q−1)/16 x^{−k}
    for j :=0 to 3 do
        c ←⊥
        while c =⊥ and nbQueries ≤ MaxQueries do
            l ← (l_0, l_1, l_2, l_3) ←R [−4, 3]^4
            b ← ZeroMatrix(8)
            for i := −4 to 3 do
                l[j] ← i
                b[i] ← O(k, l)
            end for
            c ← FindS(b)
        end while
        s[k + nj] = c
    end for
end for
Return:s
```

4.4.3 Reproducing the results of Bauer et al.

To show that their attack works, Bauer et al. provide a MagmaCAS implementation[1], which uses their own implementation of NewHope. In total they ran 1000 experiments and were able to recover 95% of the coefficients of the secret keys, which took them in average 16 700 queries per key. One goal of this work was to see if their results can be reproduced. But in difference to the code published by Bauer et al. the attack should be tested against the official reference implementation of NewHope that was submitted to the NIST-competition. Since this reference implementation is written in C, the attack also had to be ported to C as part of this thesis [42]. However, the description from the paper and the MagmaCAS implementation were used as reference.

Except for one problem, which is mentioned later in section 4.4.3, it was possible to recover all coefficients that are within the recoverable range of $[-6, 4]$, which means approximately 98% of all coefficients. This came with a down side as Figure 4.4 shows, it took in average approximately 18 900 queries, which is 2 200 queries more than Bauer et al. used. However, if the higher rate of found coefficients and the higher number of requests are combined, it can be seen that with an average of 18 queries per coefficient we only need one more. A reason for this could be that we try more often to find a suitable quadruple l until we give up and move on to the next coefficient.

[1]The MagmaCAS code can be found at https://www.di.ens.fr/~mrossi/

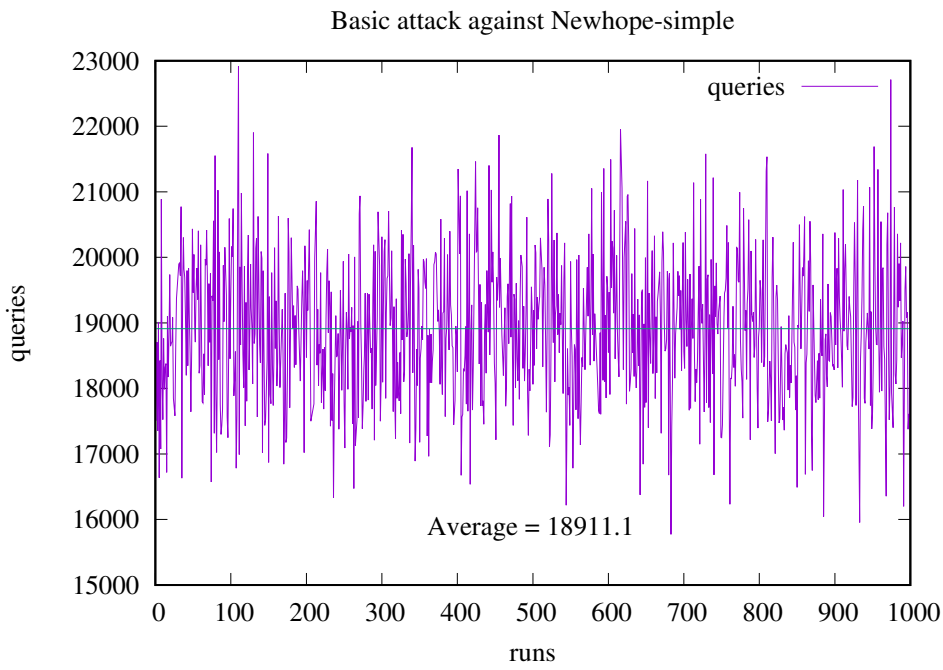


Figure 4.4: Queries with 1000 runs with the C-reference implementation

Problems

During the porting of the the attack, it was noticed that the MagmaCAS implementation differs from the formal description of the NewHope scheme. This difference is in the *Decompress* algorithm, while in the formal description and also in the C-reference implementation the following term is rounded normally, the MagmaCAS implementation always rounds down.

NewHope scheme		MagmaCAS implementation
$\mathbf{m}'[i] \leftarrow \lceil \frac{q \cdot \mathbf{c}[i]}{8} \rfloor$	vs.	$\mathbf{m}'[i] \leftarrow \lfloor \frac{q \cdot \mathbf{c}[i]}{8} \rfloor$

Although this is only a small detail, it does have consequences on the success rate of the attack, as Figure 4.5 and Example 4.7 show.

Example 4.7. *Let the elements* (l_1, l_2, l_3) *of the quadruple* l *be* $(-3, -4, 3)$ *and the attack public key will be set according to Equation 4.6,* $\mathbf{u} = \lfloor \frac{q-1}{16} \rceil \, x^{-0} = 768$*. Target index is* $s_0 = 3$*, with* $s_1 = 0, s_2 = -3, s_3 = 2$*. Then the oracle output with the floor rounding assumed by Bauer et al., will be:*

l_0	-4	-3	-2	-1	0	1	2	3
$\mathcal{O}$	+	+	+	+	+	−	+	+

So $\tau_1 = 0$ *and* $\tau_2 = 2$ *resulting in the following guess for* s_0*:*

$$\tau_1 + \tau_2 \mod 2 = 0 \implies s'_0 = \tau_1 + \tau_2 = 2 \neq 3 = s_0$$

Thus the problem is the result of $l_0 = 2$*, which should be negative, because then:*

$$\tau_1 + \tau_2 \mod 2 = 1 \implies s'_0 = 2 \left\lfloor \frac{\tau_1 + \tau_2}{2} \right\rfloor + 1 = 3 = s_0$$

To understand the positive output, it is necessary to take a closer look at the calculation that caused the output. Especially the calculation of the first bit of ν_A*. Here the term in { } denotes the difference to the values, computed when using only the floor rounding as assumed by Bauer et al.*

$$\begin{array}{rcl} \hat{\mathbf{c}}[0] & = & 6 \\ \hat{\mathbf{c}}[256] & = & 1 \\ \hat{\mathbf{c}}[512] & = & 0 \\ \hat{\mathbf{c}}[768] & = & 7 \end{array}, \quad Decompress(\hat{\mathbf{c}}) = \begin{array}{rcll} \mathbf{c}[0] & = & 9217 & \{-1\} \\ \mathbf{c}[256] & = & 1536 & \\ \mathbf{c}[512] & = & 0 & \\ \mathbf{c}[768] & = & 10753 & \{-1\} \end{array}$$

So the $\mathbf{k}'$ *will be:*

$$\begin{array}{rcrcrcrcll} \mathbf{k}' & = & \mathbf{c} & - & \mathbf{u} & \cdot & \mathbf{s} & = & \mathbf{k}' & \\ \mathbf{k}'[0] & = & 9217\{-1\} & - & 768 & \cdot & 3 & \mod q = & 6913 & \{-1\} \\ \mathbf{k}'[256] & = & 1536 & - & 768 & \cdot & 0 & \mod q = & 1536 & \\ \mathbf{k}'[512] & = & 0 & - & 768 & \cdot & -3 & \mod q = & 2304 & \\ \mathbf{k}'[768] & = & 10753\{-1\} & - & 768 & \cdot & 4 & \mod q = & 9217 & \{-1\} \end{array}$$

The values for the computation for the first bit of ν_A *will be:*

$$\left\lfloor \frac{q}{2} \right\rfloor = 6144$$

$$\sum \left\{ \begin{array}{rcrcl} |6913 - 6144| & \{-1\} & = & |769| & \{-1\} \\ |1536 - 6144| & & = & |-4608| & \\ |2304 - 6144| & & = & |-3840| & \\ |9217 - 6144| & \{-1\} & = & |3073| & \{-1\} \end{array} \right\} = 12290\{-2\}$$

So $\nu[0]$ *will be a 1, as* $12290 > q$ *or analog for the oracle positive, as* $12290 - q > 0$*, but if the* -2 *is applied,* $12288 < q$ *would result in a 0, respectively* $12288 - q < 0$ *in a negative oracle output.*

The general effect on the success rate of the attack is shown by Figure 4.5. It can be seen that only 86.1 % of all coefficients and 87% of the coefficients in $[-6, 4]$ can be recovered.

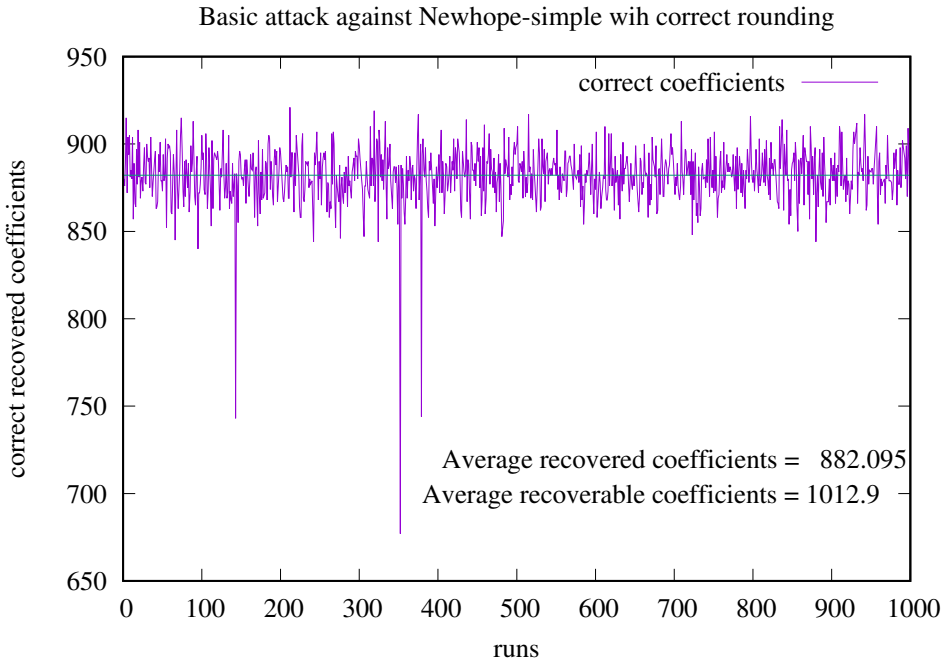


Figure 4.5: The number of absolute recovered coefficients with the Bauer et al. method, with correct rounding in the *Decompress* algorithm.

Since the *Decompress* algorithm is also included in the key mismatch oracle from Definition 4.5, the oracle must be updated. To do so it has to be formally demonstrated why it makes a difference if you round with $\lceil x \rfloor = \lfloor x + \frac{1}{2} \rfloor$ or only with down rounding $\lfloor x \rfloor$. First the changes within the *Decompress* algorithm for single

coefficient must be considered. The ciphertext is chosen according to Equation 4.3.

$$\begin{aligned}
\mathit{Decompress}(\hat{\mathbf{c}}[j]) &= \left\lceil \frac{(l_j + 4 \mod 8) \cdot q}{8} \right\rfloor \\
\text{As } l_j \in [-4,3] &\iff l_j + 4 \mod 8 = l_j + 4 \\
&= \left\lceil (l_j + 4)\, \frac{q}{8} \right\rfloor \\
&= \left\lceil (l_j + 4) \left(\frac{q-1}{8} + \frac{1}{8} \right) \right\rfloor \\
&= \begin{cases} (l_j + 4)\, \frac{q-1}{8} & \text{if } l_j \in [-4,-1] \\ (l_j + 4)\, \frac{q-1}{8} + 1 & \text{if } l_j \in [0,3] \end{cases}
\end{aligned}$$

For demonstration reasons it is assumed that all four elements of l are in $[0,3]$, so the largest effect can be seen. Therefore only the last case is used and set into the computation of the *Sign* function.

$$\begin{aligned}
\mathcal{O}' &= \mathit{Sign}\left(\sum_{j=0}^{3} \left| (l_j + 4)\ fracq - 18 + 1 - \frac{q-1}{8} s_j - \left\lfloor \frac{q}{2} \right\rfloor \right| - q \right) \\
&= \mathit{Sign}\left(4 + \sum_{j=0}^{3} \left| (l_j + 4)\ fracq - 18 - \frac{q-1}{8} s_j - \left\lfloor \frac{q}{2} \right\rfloor \right| - q \right) \\
&= \mathit{Sign}\left(4 + \sum_{j=0}^{3} \left| (l_j + 4 - s_j - 4)\, \frac{q-1}{8} \right| - q \right) && \text{as } \left\lfloor \frac{q}{2} \right\rfloor = \frac{q-1}{2} \\
&= \mathit{Sign}\left(4 + \sum_{j=0}^{3} |(l_j + s_j)| - 8 \right) && \text{as } \frac{8q}{q-1} \approx 8.0007
\end{aligned}$$

As it turns out, there might be an extra factor before the sum if the rounding is done as specified in NewHope. Since the +4 in front of the sum only represents the case where all l is in $[0,3]$, the factor in front of the sum is mostly smaller. It can be described with the function r.

Definition 4.8. Rounding factor function
Let l be a quadruple.

$$r : [-4,3]^4 \mapsto [0,4]; \qquad r(l) = \sum_{j=0}^{3} \begin{cases} 1 & l_j \geq 0 \\ 0 & l_j < 0 \end{cases}$$

Definition 4.9. Corrected key mismatch oracle
Assumed Hypothesis 4.4 is verified. Let $k \in [0, n-1]$ *be the target index,* $\hat{\mathbf{c}}$ *set as described in equation 4.3, let* l *be the integer quadruplet in* $[-4, 3]^4$ *and* r *the function from Definition 4.8. The corrected key mismatch oracle is:*

$$\mathcal{O}'(k, l) = Sign\left(r(l) + \sum_{j=0}^{3} |l_j - \mathbf{s}[k + nj]| - 8\right)$$

Despite the fact that the success rate is lower with the corrected oracle, the rest of the work it is assumed that all coefficients in the interval $[-6, 4]$ can be recovered. This was done to find possible ways to recovers the remaining coefficients in $\{-8, 7, 5, 6, 7, 8\}$. It seems likely that it should be possible to increase the success rate again with the corrected oracle.

Chapter 5
Attack Improvements

This chapter is about improving the basic attack described in chapter 4 beginning with the presentation of an approach published by Qin et al. [36] and then discuss the problems discovered during the reproduction process of this approach. Furthermore, the improvements developed in the scope of this work are presented. For instance, it was possible to reduce the number of oracle queries needed for the attack. The more significant improvement, however, is the recovery of additional coefficients and the narrowing of coefficients to speed up the subsequent brute force search.

5.1 Proposals from Qin et al.

After Bauer et al. has published their attack Qin et al. published «A Complete and Optimized Key Mismatch Attack on NIST Candidate NewHope» [36]. They also tried to reproduce the results of Bauer et al., but were only able to reproduce about 73% of the 1024 coefficients of $\mathbf{s}$. As the code is not published and it is not explained in detail what exactly led to the lower success rate, only assumptions can be made. It seems likely that this is partly caused by the rounding problem described in section 4.4.3. It is quite possible that there were other problems, as they found 13 % less coefficients than when only correcting the rounding problem. An indication for this is that they found a fourth case additionally to the three cases described in section 4.4.2, when l_0 is iterated from -4 to 3. They call it also an «favorable case». The oracle output is at the beginning negative, then becomes positive and then negative again, as shown below:

l_0	-4	-3	-2	-1	0	1	2	3
$\mathcal{O}$	$-$	$-$	$-$	$+$	$+$	$+$	$-$	$-$

However, this fourth case could not be detected in the context of this work and no further attempt was made to reproduce their method of recovering the coefficients in

$[-6.4]$. The more interesting part of the work from Qin et al. is that they proposed a smarter way to recover the coefficients in $\{-8, -7, 5, 6, 7, 8\}$.

They analyzed the tuples $(s[i], s[i+256], s[i+512], s[i+768]) : i \in [0, 255]$ of 10^6 secret keys. The results show that 4.16% of the tuples have at least one coefficient that is not in $[-6, 4]$ and 98.5% of these tuples have only one coefficient that is not in $[-6, 4]$. This means that all tuples with three or four coefficients in $[-6, 4]$ represent 99.94% of all coefficients. Therefore Qin et al. have concentrated on the coefficients that are in such a tuple.

As used in the description of the attack from Bauer et al. in subsection 4.4.2 the term $s_j : j \in [0, 4]$ is used for the tuple $(s[i], s[i+256], s[i+512], s[i+768]) : i \in [0, 255]$ of Alice's secret key $\mathbf{s}$. These four coefficients are involved in decoding the bit $\nu_A[i]$ of Alice's key string. The unknown coefficient will be the first of the tuple s_0, meaning that $s_1, s_2, s_3 \in [-6, 3]$ are already recovered. The procedure is analogous if one of the other coefficients s_1, s_2, s_3 is the unknown. The idea of the improvement is to extract the sum $m = \sum_{j=0}^{3} |s_j|$ of all coefficients s_j, with only s_0 unknown, which can be calculated afterwards by $s_0 = m - \sum_{j=1}^{3} |s_j|$. To find out m, the attacker $\mathcal{A}$ has to choose his public parameter as follows:

$$\mathbf{s}' = \mathbf{e}'' = 0, \tag{5.1}$$

$$\mathbf{u} = \mathbf{a}\mathbf{s}' + \mathbf{e}' = \mathbf{e}' = h \cdot x^{512} : h \in [0, q-1] \tag{5.2}$$

$$\nu_{\mathcal{A}} = (0, \ldots, 0) \text{ except at the target index } \nu_{\mathcal{A}}[i] = 1 \tag{5.3}$$

This means that the ciphertext will be just $\mathbf{k}$, which is be set according to the key string $\nu_{\mathcal{A}}$ from Equation 5.3:

$$\begin{aligned}
\mathbf{c} &= \mathbf{b}\mathbf{s}' + \mathbf{e}'' + \mathbf{k} \\
&= \mathbf{k} \\
&= \textit{Key Encode}(\nu_{\mathcal{A}}) \\
&= \left\lfloor \frac{q}{2} \right\rfloor x^i + \left\lfloor \frac{q}{2} \right\rfloor x^{i+256} + \left\lfloor \frac{q}{2} \right\rfloor x^{i+512} + \left\lfloor \frac{q}{2} \right\rfloor x^{i+768}
\end{aligned}$$

According to that the compressed ciphertext is:

$$\hat{\mathbf{c}} = \textit{Compress}(\mathbf{c}) = 4x^i + 4x^{i+256} + 4x^{i+512} + 4x^{i+768}$$

These values are then used in the combination with the key mismatch oracle $\mathcal{O}_2$ defined in Definition 4.3 to find the sum m. Therefore the attacker iterates h from zero to $q - 1$. In the beginning the oracle output will be positive and changes

eventually to a negative one. The value of h at which the output changes can then be used by the attacker to calculate $m = \lfloor \frac{q}{h} \rceil$. The full procedure of finding m in illustrated in Algorithm 5.1. After the attacker has determined m he is able to calculate the value of s_0 using s_1, s_2, s_3.

$$s_0 = m - |s_2| - |s_2| - |s_3|$$

Algorithm 5.1 Find-m

Input: Target index $i \in [0, 255]$

```
for h := 1 to q − 1 do
    u[512] ← h
    ν_A = (0, ..., 0) and ν_A[i] = 1
    k ← Key Encode(ν_A)
    ĉ ← Compress(k)
    r ← O_2(u, ĉ, ν_A)
    if r < 0 then
        m = ⌊q/h⌉
        break
    end if
end for
```

Return: m

Proof To understand why the attacker can calculate m using h, the decapsulation procedure of Alice must be inspected. Therefore we use the values defined in Equation 5.1 and Equation 5.2, beginning with the ciphertext $\mathbf{c}'$.

$$\begin{aligned}\mathbf{c}' = \mathit{Decompress}(\hat{\mathbf{c}}) &= \left\lfloor \frac{q}{2} \right\rfloor x^i + \left\lfloor \frac{q}{2} \right\rfloor x^{i+256} + \left\lfloor \frac{q}{2} \right\rfloor x^{i+512} + \left\lfloor \frac{q}{2} \right\rfloor x^{i+768} \\ &= 6145x^i + 6145x^{i+256} + 6145x^{i+512} + 6145x^{i+768}\end{aligned}$$

As $x^{1024} \equiv -1$ in $\mathcal{R}_q$ meaning $3x^{1025} \equiv -3x \in \mathcal{R}_q$, the coefficients in $\mathbf{k}'$ are:

$$\begin{aligned}\mathbf{k}' &= \mathbf{c} - \mathbf{s} \cdot \mathbf{u} \\ \mathbf{k}'[i] &= 6145 - (-s_2 \cdot h) \mod q \\ \mathbf{k}'[i+256] &= 6145 - (-s_3 \cdot h) \mod q \\ \mathbf{k}'[i+512] &= 6145 - s_0 \cdot h \mod q \\ \mathbf{k}'[i+768] &= 6145 - s_1 \cdot h \mod q\end{aligned}$$

Then the *Key Decode*($\mathbf{k}'$) computation will result in:

$$
\begin{aligned}
t = & \sum_{j=0}^{3} |k[i+256j] - \left\lfloor \frac{q}{2} \right\rfloor| \\
t = & \quad |6145 + s_2 \cdot h - 6144| \\
& + |6145 + s_3 \cdot h - 6144| \\
& + |6145 - s_0 \cdot h - 6144| \\
& + |6145 - s_1 \cdot h - 6144| \\
= & |1 + s_2 \cdot h| + |1 + s_3 \cdot h| + |1 - s_0 \cdot h| + |1 - s_1 \cdot h| \\
= & 1 + s_2 \cdot h + 1 + s_3 \cdot h + s_0 \cdot h - 1 + s_1 \cdot h - 1 \\
= & (s_2 + s_3 + s_0 + s_1) \cdot h \\
= & m \cdot h
\end{aligned}
$$

According to the definition of *Key Decode* it will be checked if t is greater than or less than q. Which means that at the beginning with $h = 0$ implying $m \cdot h < q$ *Key Decode* outputs a one, resulting in a positive output from $\mathcal{O}_2$. If the oracle output is negative then it means that *Key Decode* outputs a zero and $m \cdot h \approx q$, so m can be calculated with the h where the oracle output changes from positive to negative.

$$m \cdot h \approx q \Leftrightarrow m = \left\lfloor \frac{q}{h} \right\rceil \qquad \square$$

Qin et al. have done some experiments, but they did not mention which language was used or published any code. The number of queries their needed for there attack is fifty times higher than what Bauer et al. used, instead of 16 700 Qin et al. used in average 879 725 queries. One reason might be that more coefficients are recovered by sending queries to the oracle instead of brute forcing them, but also because of their way of finding the coefficients in $[-6, 4]$ needs more queries. This is the reason why only the approach from Bauer et al. for recovering the coefficients in $[-6, 4]$ is presented in this work.

5.2 Problems in the Qin et al. approach

As with the attack of Bauer et al., the aim of this work was to practically reproduce the improvement from Qin et al. Only the formal description of their publication could be used [36], since the source code of their experiments is not published. Again the attack was implemented in C and is tested against the reference implementation that was submitted by the authors of NewHope to the NIST-Competition. Unfortunately, it was not possible to reproduce the published results, caused by two problems, which were discovered during the reproduction process. Both have the consequence that the improvement of the attack as proposed by Qin et al. does not work successfully.

Multiplication of u and s

Similar to the basic attack by Bauer et al., an important condition for the improvement is that the key string decapsulated by Alice differs from the key string the attacker assumes only in one bit. In this case means that Alice's key string is either as assumed in Equation 5.3 or consists only of zeros, $\nu_A = (0, \ldots, 0)$, limiting the only bit that can vary to $\nu[i]$. However this is not always the case in the approach from Qin et al.. To understand why, example 5.1 is given.

Example 5.1. *Assume that* $\mathbf{u}$ *and* $\mathbf{c}$ *are set as specified in Equation 5.2, with the target index i being zero, accordingly the key string is* $\nu_{\mathcal{A}} = (1, 0, \ldots, 0)$*. For this example all the coefficients of the targeted secret key* $\mathbf{s}$ *are zero except of the following eight:*

$$\begin{array}{llll} s[0] = 6 & s[256] = 1 & s[512] = 0 & s[768] = -4 \\ s[1] = 4 & s[257] = 4 & s[513] = 6 & s[769] = 6 \end{array}$$

Correspondingly, the sum m is:

$$\begin{aligned} m &= |s[0]| + |s[256]| + |s[512]| + |s[768]| \\ &= |6| + |1| + |0| + |-4| \\ &= 11 \end{aligned}$$

To determine the key string that is decoded by Alice one has to look at the Key Decode$(\mathbf{k}')$ *algorithm for the index 0. As described above, the output is based on whether* $m \cdot h$ *is greater than or less than q. At the beginning the h will be zero, so:*

$$m \cdot h = 6 \cdot 0 = 0 < q \Rightarrow \textit{Key Decode}(\mathbf{k}')[0] = 1$$

This will result in the wanted positive oracle output. With h increasing, the output will get positive when Key Decode$(\mathbf{k}')$*[0] changes to 0, which will be the case if* $h = 1118$ *as:*

$$m \cdot h = 6 \cdot 1118 = 12298 > q \Rightarrow \textit{Key Decode}(\mathbf{k}')[0] = 0$$

As described in Algorithm 5.1, the first time there is a negative oracle output, no further queries are made and $m = \left\lfloor \frac{q}{h} \right\rceil$ *gets calculated with the most recent h. Assuming that all other bits of* ν_A *are still zero, which is not the case for* $\nu_A[1]$*.*

To understand why $\nu_A[1]$ also changes to a one, first the relevant coefficients of $\mathbf{k}'$ need to be calculated.

$$\begin{aligned}
\mathbf{k}' &= \mathbf{c} - \mathbf{s} \cdot \mathbf{u} \\
\mathbf{k}'[1] &= 0 - (-s[513] \cdot h) \mod q &&= s[513] \cdot h \\
\mathbf{k}'[257] &= 0 - (-s[769] \cdot h) \mod q &&= s[769] \cdot h \\
\mathbf{k}'[513] &= 0 - s[0] \cdot h \mod q &&= -s[0] \cdot h \\
\mathbf{k}'[769] &= 0 - s[257] \cdot h \mod q &&= -s[257] \cdot h
\end{aligned}$$

These coefficients are used within the Key Decode algorithm, which will use the sum t to the decoded bit Key Decode($\mathbf{k}'$)[1]:

$$\begin{aligned}
t &= \sum_{j=0}^{3} |\mathbf{k}'[1 + 256j] - \lfloor \frac{q}{2} \rfloor| \\
&= \left| s[513] \cdot h - \left\lfloor \frac{q}{2} \right\rfloor \right| \\
&\quad + \left| s[769] \cdot h - \left\lfloor \frac{q}{2} \right\rfloor \right| \\
&\quad + \left| -s[0] \cdot h - \left\lfloor \frac{q}{2} \right\rfloor \right| \\
&\quad + \left| -s[257] \cdot h - \left\lfloor \frac{q}{2} \right\rfloor \right| \\
&= |4h + 4h + 6h + 6h - 2q| \\
&= |20h - 2q|
\end{aligned}$$

For $h = 0$, the sum t will be $2q$ resulting in a zero as output. As h increases the sum will decrease and for $h = 615$, t will be:

$$\begin{aligned}
t &= |20h - 2q| \\
&= |20 \cdot 615 - 2q| \\
&= |12300 - 2q| \\
&= q - 11
\end{aligned}$$

Obviously $q - 11$ is smaller than q and Key Decode outputs a one, resulting the bit $\nu_A[1]$ of Alice's key string to be one. As h gets iterated from 0 to $q - 1$ the bit $\nu_A[1]$ changes earlier than the targeted bit $\nu_A[0]$. Therefore the oracle output will change at $h = 615$ instead of $h = 1118$ so a wrong m and correspondingly a wrong s_0 will be calculated by $\mathcal{A}$.

Although the example 5.1 shown above is somewhat artificial, reproducing the attack has shown that the problem occurs with almost every secret key $\mathbf{s} \xleftarrow{R} \psi_8^N$.

Regarding the fact that it is very likely that other bits of the key string change earlier than intended, the attack is impracticable.

Absolute value in *Key Decode*

In addition to the problem previously described, another problem occurred during the reproduction. It arises in the *Key Decode* algorithm, where the coefficients of $\mathbf{k}'$ get interpreted as normal integers instead of elements of $\mathbb{Z}_q$ Unfortunately, the exact cause could not be identified since no source code was published by Qin et al. However, the problem will be illustrated by the following concrete example.

Example 5.2. *Let the coefficients of the secret key* $\mathbf{s}$ *be* $(s_0 = 6, s_1 = 1, s_2 = 3, s_3 = 1)$*, so the coefficient to recover is* s_0 *and the rest is already known, as they could be recovered with the basic attack from Bauer et al. described in subsection 4.4.2. According to the coefficients, the targeted sum* m *is* 11*, which means that the first* h *at which the oracle output changes is 1118, since:*

$$m \cdot 1118 = 12298 > q$$

Accordingly, we will look at the individual steps the decapsulation process done by Alice, starting with the multiplication of $\mathbf{u}$ *and* $\mathbf{s}$*.*

$$\begin{array}{llll}
\mathbf{s}\cdot\mathbf{u}[0] & = -s_2h \mod q & = -3h \mod q & = -3354 \mod q & = 8935 \\
\mathbf{s}\cdot\mathbf{u}[256] & = -s_3h \mod q & = -h \mod q & = -1118 \mod q & = 11171 \\
\mathbf{s}\cdot\mathbf{u}[512] & = s_0h \mod q & = 6h \mod q & = 6708 \mod q & = 6708 \\
\mathbf{s}\cdot\mathbf{u}[768] & = s_1h \mod q & = h \mod q & = 1118 \mod q & = 1118
\end{array}$$

These values are then used to calculate the polynomial $\mathbf{k}' = \mathbf{c} - \mathbf{us}$*, note* $\lfloor \frac{q}{2} \rfloor = 6144$*:*

$$\begin{array}{llll}
\mathbf{k}'[0] & = 6144 - 8935 \mod q & = -2791 \mod q & = 9498 \\
\mathbf{k}'[256] & = 6144 - 11171 \mod q & = -5027 \mod q & = 7262 \\
\mathbf{k}'[512] & = 6144 - 6708 \mod q & = -564 \mod q & = 11725 \\
\mathbf{k}'[768] & = 6144 - 1118 \mod q & = 5026 \mod q & = 5026
\end{array}$$

Finally Key Decode$(\mathbf{k}')$ *uses these coefficients to decode* ν_A*, so the sum* t *has to be calculated:*

$$
\begin{aligned}
t = \quad & \textstyle\sum_{j=0}^{3} |\mathbf{k}'[0+256j] - \lfloor \frac{q}{2} \rfloor| \\
= \quad & |9498 - \lfloor \tfrac{q}{2} \rfloor| && = |3354| && = 3354 \\
& +|7262 - \lfloor \tfrac{q}{2} \rfloor| && = |1118| && = 1118 \\
& +|11725 - \lfloor \tfrac{q}{2} \rfloor| && = |5518| && = 5518 \\
& +|5026 - \lfloor \tfrac{q}{2} \rfloor| && = |-1118| && = 1118 \\
& && = 11171 < q \Rightarrow \textit{Key Decode}(\mathbf{k}')[0] = 0
\end{aligned}
$$

Obviously there is a problem as $t = 11171 \neq 12298 = m \cdot h = 11 \cdot 1118$
This could be fixed if the third coefficient is changed to its a negative representation in $\mathbb{Z}_q$ $5518 \mod q \equiv -6708 \mod q$ *then the sum* t *is:*

$$t = |3354| + |1118| + |-6708| + |-1118| = 12298 = m \cdot h > q \Rightarrow \textit{Key Decode}(\mathbf{k}')[0] = 1$$

As example 5.2 demonstrates, the cause of the problem is of a more technical nature. The values of the coefficients of the polynomials in $\mathcal{R}_q$ are in $\mathbb{Z}_q$. The reference implementation uses *uint16* to store all coefficients so they can only be represented by numbers in $[0, 2^{16} - 1]$. For NewHope this means that all coefficients are always represented with a number in $[0, q-1]$. The reason why this causes a problem can be reduced to the fact that the following applies:

$$-6 \cdot h \mod q = -6708 \mod q \equiv 5518 = q - 6 \cdot h \text{ in the group } \mathbb{Z}_q$$

But for the absolute values:

$$|-6 \cdot h| \neq |q - 6 \cdot h| = |-6 \cdot h \mod q| \text{ even when } 6 \cdot h < q$$

The parallel occurrence of the two described problems is the reason why the improvement of the attack of Qin et al. could not be practically reproduced. The problem with the absolute value is probably caused by a different technical implementation of the NewHope scheme from Qin et al, than the official reference implementation. Therefore a key element to be able to discover both problems, were that the attack was re-implemented and tested as part of this work. The version of the source code that implements the improvement by Qin et al. and demonstrates both problems is published in [43].

5.3 Boundary check to reduce the amount of oracle queries

Additional to the extension of the interval in which coefficients can be recovered described in section 5.1 and section 5.4, the attack also can be improved by reducing the amount of oracle queries needed. This is relevant because the fewer queries are

needed for a successful attack, the faster an attack can be executed. This means that the attack will work even if Alice/a server only reuses its key for a shorter period. So it can be said that the fewer queries are needed, the more practical the attack is.

There are several ways how the number of queries can be reduced. The first is not described by Bauer et al. in their paper [4], but is applied in the code provided. It is about an early detection of whether one gets a favorable case (iii) as described in subsection 4.4.2 or not, to eventually chose a new $l \xleftarrow{R} [-4, 3]^4$. Short recap, the quadruple l is used to create the attack ciphertext.

$$\hat{\mathbf{c}} = \sum_{j=0}^{3} ((l_j + 4) \bmod 8) \cdot x^{256j}$$

By choosing the ciphertext this way it is possible to define the key mismatch oracle $\mathcal{O}(k, l)$ from Definition 4.5 that has either a positive or negative output. A favorable case occurs, if over the iteration of l_i in $[-4, 3]$ the oracle output is positive at the beginning, then becomes negative and at the end becomes positive again, as illustrated in Table 5.1. Since in all other cases the oracle output is negative in at

l_i	-4	-3	-2	-1	0	1	2	3
$\mathcal{O}$	+	+	+	-	-	-	+	+

Table 5.1: Oracle output in a favorable case

least one of the two boundary cases $l_i = -4$, $l_i = 3$, these two boundary cases can be checked first, instead of simply iterating l from -4 to 3 as described in Algorithm 4.2. If one of the outputs is not positive, a new $l \xleftarrow{R} [-4, 3]^4$ can be selected. Saving 6 requests if the quadruple is not suitable.

A further reduction was realised as part of this work. Instead of picking the quadruple l at random and hoping that the values will result in a favorable case, the values can be chosen more sophisticatedly. Unfortunately this will not work for s_0, s_1, s_2, as there is not enough knowledge to fully control v. To produce a favorable case as described in section 4.4.2, the sum v needs to be smaller than zero but still close to zero, as illustrated in Figure 5.1. The sum v is slightly different than in section 4.4.2, as we are now targeting s_3.

$$v = \sum_{j=0}^{2} |l_j - \frac{s_j}{2}| - 8$$

Since s_0, s_1, s_2 are already known when recovering s_3 v can already be calculated after selecting $l \xleftarrow{R} [-4, 3]$. This saves the two requests for the boundary check

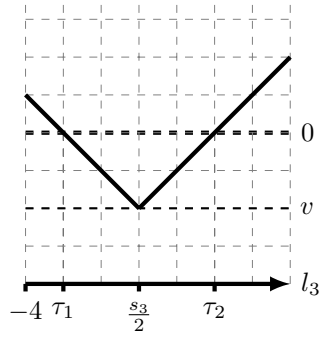


Figure 5.1: Favorable case for $s_3 = -2$ as $\tau_1 + \tau_2 = -3 + 1 = -2 \mod 0$, with $v = -2$

described above. However, it is also possible to select the values l_0, l_1, l_2 directly in such a way that v will have the intended value, with a tolerance of $+0.5$. Therefore it can be ensured that the queries will result in a favorable case. The tolerance of $+0.5$ is because of $\frac{s_j}{2}$ might not be an integer values, but for l_j only integer values can be chosen. For this reason the Algorithm 5.2 uses in line three $v_r + 0.5$ instead of v_r, which results in the tolerance -0.5. Even though l_j could also be -4, it simplifies the code to just use 3 in the minimum function and limit the $l_j \in [-3, 3]$.

Algorithm 5.2 Set-l-with-v

Input: $s_1, s_2, s_3 \in [-6, 4]$, target sum v_t

1: $v_r \leftarrow v_t + -\left|\frac{s_1}{2}\right| - \left|\frac{s_2}{2}\right| - \left|\frac{s_3}{2}\right|$
2: **for** $j := 1$ to 3 **do**
3: $\quad l_j = Sign(s_j) \cdot \lfloor \min(v_r + 0.5, 3) \rfloor$
4: $\quad v_r \leftarrow v_r - |v_j|$
5: **end for**

Return: $l_1, l_2, l_3 \in [-3, 3]$

How the different optimizations reduce the amount of queries needed for the basic attack is shown in Table 5.2. The boundary check that was already used by Bauer et al. brings the biggest reduction of about 32%. Even though the optimization of l is not so effective, if both of them are combined the average amount of queries is reduced by 46%. All the results were produced using the the code published in [41].

no optimization	≈ 27600
set l with v	≈ 23900
boundary check	≈ 18900
boundary check + set l with v	≈ 14800

Table 5.2: The average amount of query within 1000 attack runs, with the different optimizations

5.4 Reduce search space for brute force

Since it was not possible to use the improvement from Qin et al. to recover the coefficients in $\{-8, -7, 5, 6, 7, 8\}$, the question arose whether there are other ways to recover these coefficients. The benchmark here is a brute force search with a complexity of 6^{11}. This is due to the fact that there are about 11 coefficients per secret key that are not within the recoverable interval and that there are six options $\{-8, -7, 5, 6, 7, 8\}$ for each of them. The optimal solution would be to derive all the coefficients directly from additional key-mismatch queries. Unfortunately this could not be achived for all of them, but with some additional queries it is possible to recover some more or narrowing them down to reduce the search space for the brute force search.

Like in section 5.1 and section 5.3 again the fact is used that most of the coefficients that cannot be recovered belong to a tuple (s_0, s_1, s_2, s_3), where three out of four coefficients are already known. Therefore Algorithm 5.2 is used to set l, which allows to have control over the sum v. Additionally, the control over v allows you to extract information from a non favorable case, were only one sign change occurs. In the rest of this section it is assumed that $s_1, s_2, s_3 \in [-6, 4]$ are the known coefficients and $s_0 \in \{-8, -7, 5, 6, 7, 8\}$ is the missing targeted one. First it needs to be checked whether the sum v will be an integer or has a rest of 0.5, this can be achieved by calculating the sum of the known coefficients.

$$v' = \sum_{j=1}^{3} \frac{s_j}{2} \text{ and check } v' - \lfloor v' \rfloor \stackrel{?}{=} 0$$

The division by two is caused by the extension to the full interval $[-6, 4]$ described in section 4.4.2.

The second step is to set l using algorithm 5.2 so that $v = -1.0$ respective $v = -0.5$ and use it for eight oracle queries where l_0 is iterated from -4 to 3. In the case $v' - \lfloor v' \rfloor = 0$ is an integer, eight more requests must be made with $v = 0$. The resulting outputs for the different coefficients are listed in Table 5.3. Obviously there

s_j	$v = -0.5$	$v = -1.0$	$v = 0$
-8	[− + + + + + ++]	[− − + + + + ++]	[− + + + + + ++]
-7	[− − + + + + ++]	[− − + + + + ++]	[+ + + + + + ++]
5	[+ + + + + + −−]	[+ + + + + + −−]	[+ + + + + + ++]
6	[+ + + + + + +−]	[+ + + + + + −−]	[+ + + + + + +−]
7	[+ + + + + + +−]	[− + + + + + +−]	[+ + + + + + ++]
8	[− + + + + + ++]	[− + + + + + +−]	[+ + + + + + ++]

Table 5.3: Different oracle outputs for coefficients in $\{-8, -7, 5, 6, 7, 8\}$ over the course of $l_0 \in [-4, 3]$

are no more favorable cases, but some output patterns can be identified and some coefficients can be recovered and some at least narrowed down. The colors in the table give an indication which coefficient results in a unique pattern, allowing them to be recovered or to which two options the coefficient can be narrowed down.

$v' - \lfloor v' \rfloor$	Recoverable	Narrow down
0.5	$-7, 5$	$\{-8, 8\}, \{6, 7\}$
0	$-8, -7, 5, 6$	$\{7, 8\}$

Table 5.4: The possible interpretation from query patterns of Table 5.3

As Table 5.4 shows the five is always recoverable, which is an improvement as due to the centered binomial distribution used for New Hope, the five is the most common coefficient within the here relevant interval. This thesis also included to show that the improvements are applicable in practice. Therefore both improvements were implemented in C and tested against the reference implementation of NewHope. The full C code is published in [44]. The same code was used for the analysis presented in Table 5.5.

Per secret key **s** with 1024 coefficients:	
Number of coefficients in $\{-8, -7, 5, 6, 7, 8\}$	11
Number of correct recovered coefficients	1022
Number of coefficients narrowed down to two options	1.5
Number of total unknown coefficients	0.5

Table 5.5: The average results from 1000 attacks, with our own improvement to recover more coefficients and to reduce the search space

The amount of correct coefficients increases to 1022, so only two are missing. Mostly one of these two is already determined down to two options, but often this is the cause for both. Thus the complexity for the brute force search could be reduced from $6^{11} = 2^{25}$ to 4 to $8 \cdot 2 = 16$ possibilities, which more or less could be considered as not relevant. Meaning that this improvement makes the attack more practical.

Algorithm 5.3 NarrowDownCoefficient-s_0

Input: $s_1, s_2, s_3 \in [-6, 4]$

$v' = \sum_{j=1}^{3} s_j$
$v_{int} = v' - \lfloor v' \rfloor = 0$
$l \leftarrow (-4,\ \textit{Set-l-with-v}(s_1, s_2, s_3, v_t = -1.0))$
for $j := 0$ to 7 **do** ▷ Doing the first eight oracle queries
    $r[j] \leftarrow \mathcal{O}(0, l)$
end for
if $v_{int} = 0$ **then** ▷ If v is an integer make a second run with $v = 0$
    $l \leftarrow (l_0 = -4,\ \textit{Set-l-with-v}((s_1, s_2, s_3), v_t = 0))$
    for $j := 0$ to 7 **do**
        $r_2[j] \leftarrow \mathcal{O}(0, l)$
    end for
end if
▷ After all queries, check for different patterns
if $v_{int} = 0$ and $r = [- + + + + + + +]$ **then** $s_0 \leftarrow \{-8, 8\}$
end if
if $v_{int} = 0$ and $r = [- - + + + + + +]$ **then** $s_0 \leftarrow -7$
end if
if $v_{int} = 0$ and $r = [+ + + + + + - -]$ **then** $s_0 \leftarrow 5$
end if
if $v_{int} = 0$ and $r = [+ + + + + + + -]$ **then** $s_0 \leftarrow \{6, 7\}$
end if
if $v_{int} = 0.5$ and $r = [- - + + + + + +]$ **then**
    if $r_2 = [- + + + + + + +]$ **then** $s_0 \leftarrow -8$
    else $s_0 \leftarrow -7$
    end if
end if
if $v_{int} = 0.5$ and $r = [+ + + + + + - -]$ **then**
    if $r_2 = [+ + + + + + + +]$ **then** $s_0 \leftarrow 5$
    else $s_0 \leftarrow 6$
    end if
end if
if $v_{int} = 0$ and $r = [- + + + + + + -]$ **then** $s_0 \leftarrow \{7, 8\}$
end if

Return: s_0

Chapter 6
Summary and Conclusion

This work is part of the process to develop the new type of post-quantum (PQ) cryptography. To understand why this new type of cryptography is needed, the thesis starts with a simplified description of quantum computers and their threats for the currently used cryptography. A major part of this process is the competition launched by the National Institute of Standards and Technology (NIST) to find a PQ scheme suitable to be used on the internet. We present the lattice-based Key Encapsulation mechanism NewHope that was submitted to the first round of the NIST-competition and got simplified for the second round. Our contribution is to analyse a published attack approach against NewHope and develop improvements and all of this is not only done as theoretical work but also practically evaluated by implementing it and test it against the official C-reference implementation of NewHope that is written by the authors. All different versions of the produced source code are published in [43], [42], [41], [44].

The main focus lies on the key reuse attack from Bauer et al. [4] and the improvement published by Qin et al. [36]. The key element of this analysis is to reproduce the attack and test it against the C-reference implementation that was submitted to NIST-competition by the authors of NewHope. This makes it possible to detect a problem in the approach of Bauer et al., which is caused by a small technical detail where their implementation differed from the reference implementation. Instead of the normal rounding function $\lfloor \ \rceil$ they only use floor rounding $\lfloor \ \rfloor$ in the *Key Decode* algorithm. The full reason of this problem was discovered late in the course of this thesis, so due to the lack of time it was not possible to develop a solution and reach the success rate previously achieved by Bauer et al.. Still though it is possible to more or less reproduce their results and to recover the most of the coefficients of Alice's secret key.

Unfortunately the same cannot be said about the improvement from Qin et al., as we discovered two problems in their approach. The first is more of a mathematical nature and caused by an incorrect assumption in the multiplication of two polynomials,

while the second one is again a technical one, since it might be caused by an inaccurate representation of elements in the group $\mathbb{Z}_q$.

As the improvement from Qin et al. does not work for us, we developed an new way to increase the amount of recoverable coefficients. With our approach it is possible to recover more then 99% of the coefficients. Another optimization is done to reduce the amount of queries needed for the basic attack by using the already known coefficients.

Despite the success of being able to recover nearly hundred percent of the secret key the attack can yet not be considered as critical for NewHope. Although it is possible that there will maybe implementations that use the private key for a longer period of time, the assumptions made for the attack are still too strict. The authors of NewHope have already recommended to generate new private keys for each key exchange when introducing the procedure. Furthermore the attack only works against the CPA-secure scheme of NewHope and not against the more secure CCA version, which would be the one used later. So the next step for further work is to develop a key mismatch oracle based on the CCA version of NewHope. A possible way of doing this could for example to check if a timing attack can be used to create a key mismatch oracle for the CCA-secure version of NewHope.

Overall one important advice can be given as result of this work: always test an attack against the reference code if it is available. It reduces the number of possible error sources and thus ensures that the scheme is implemented correctly. Personally, I am very curious to see if a lattice-based approach will prevail in the NIST-competition and what other attack approaches against NewHope will be published in the future.

References

[1] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 601–610. ACM, 2001.

[2] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Newhope without reconciliation. *IACR Cryptology ePrint Archive*, 2016:1157, 2016.

[3] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 327–343. USENIX Association, 2016.

[4] Aurélie Bauer, Henri Gilbert, Guénaël Renault, and Mélissa Rossi. Assessment of the key-reuse resilience of newhope. In Mitsuru Matsui, editor, *Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings*, volume 11405 of *Lecture Notes in Computer Science*, pages 272–292. Springer, 2019.

[5] Mihir Bellare, Dennis Hofheinz, and Eike Kiltz. Subtleties in the definition of IND-CCA: when and how should challenge decryption be disallowed? *J. Cryptology*, 28(1):29–48, 2015.

[6] Daniel J Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.

[7] Daniel J. Bernstein, Nadia Heninger, Paul Lou, and Luke Valenta. Post-quantum RSA. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 311–329. Springer, 2017.

[8] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The making of KECCAK. *Cryptologia*, 38(1):26–60, 2014.

[9] Matt Braithwaite. Experimenting with post-quantum cryptography. https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html, 2016. [Accessed: 2020-04-08].

[10] Lily Chen, Dustin Moody, and Yi-Kai Liu. Call for proposals on post-quantum cryptography, 2016. https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals [Accessed:2019-10-24].

[11] Joan Daemen and Vincent Rijmen. Rijndael for AES. In *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*, pages 343–348. National Institute of Standards and Technology„ 2000.

[12] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[13] Jintai Ding, Saed Alsayigh, R. V. Saraswathy, Scott R. Fluhrer, and Xiaodong Lin. Leakage of signal function with reused keys in RLWE key exchange. In *IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25, 2017*, pages 1–6. IEEE, 2017.

[14] Nadia Heninger Aurore Guillevic Pierrick Gaudry Fabrice Boudot Emmanuel Thomé, Paul Zimmermann. Factorization of rsa-250, 2020. https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html [Accessed: 2019-03-16].

[15] Scott R. Fluhrer. Cryptanalysis of ring-lwe based key exchange with key share reuse. *IACR Cryptology ePrint Archive*, 2016:85, 2016.

[16] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[17] Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. Software speed records for lattice-based signatures. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, volume 7932 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2013.

[18] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 789–815, 2016.

[19] Matthew Hayward. Quantum computing and Shor's algorithm. *Sydney: Macquarie University Mathematics Department.*, 2008.

[20] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors,

Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, 2017.

[21] Éliane Jaulmes and Antoine Joux. A chosen-ciphertext attack against NTRU. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2000.

[22] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350. Springer, 2010.

[23] Hendrik Willem Lenstra, Arjen K Lenstra, L Lovfiasz, et al. Factoring polynomials with rational coeficients. *Mathematische Annalen*, 1982.

[24] Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. In Ronald Cramer, editor, *Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, Barcelona, Spain, March 9-12, 2008. Proceedings*, volume 4939 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 2008.

[25] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

[26] Jacques Martinet. *Perfect lattices in Euclidean spaces*, volume 327. Springer Science & Business Media, 2013.

[27] John Martinis and Sergio Boixo. Quantum supremacy using a programmable superconducting processor, 2019. https://ai.googleblog.com/2019/10/quantum-supremacy-using-programmable.html [Accessed: 2019-10-24].

[28] Daniele Micciancio and Oded Regev. *Lattice-based Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[29] Jörn Müller-Quade. Hieroglyphen, Enigma, RSA-Eine Geschichte der Kryptographie. *Fakultät für Informatik der Universität Karlsruhe. Abgerufen*, 28, 2008.

[30] Michael A. Nielsen and Isaac Chuang. Quantum computation and quantum information. *American Journal of Physics*, 70(5):558–559, 2002.

[31] Edwin Pednault, John Gunnels, and Jay Gambetta. On "quantum supremacy", 2019. https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/ [Accessed: 2020-02-17].

[32] Chris Peikert. Lattice cryptography for the internet. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, volume 8772 of *Lecture Notes in Computer Science*, pages 197–219. Springer, 2014.

[33] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.

[34] Klaus Pommerening. Kasiski's test: Couldn't the repetitions be by accident? *Cryptologia*, 30(4):346–352, 2006.

[35] John Preskill. Quantum computing and the entanglement frontier. *arXiv preprint arXiv:1203.5813*, 2012.

[36] Yue Qin, Chi Cheng, and Jintai Ding. A complete and optimized key mismatch attack on NIST candidate newhope. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part II*, volume 11736 of *Lecture Notes in Computer Science*, pages 504–520. Springer, 2019.

[37] E. Rescorla. The transport layer security (tls) protocol version 1.3, 2015. https://tools.ietf.org/html/draft-ietf-tls-tls13-07 [Accessed: 2019-10-30].

[38] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[39] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.

[40] Markus Swarowsky. Analysis of key reuse attacks on the post-quantum scheme newhope. Project report in TTM4502, Department of Information Security and Communication Technology, NTNU – Norwegian University of Science and Technology, Dec. 2019.

[41] Markus Swarowsky. Masterthesis code - analysis of reducing queries. https://doi.org/10.5281/zenodo.3728199, March 2020.

[42] Markus Swarowsky. Masterthesis code - Bauer attack. https://doi.org/10.5281/zenodo.3714908, March 2020.

[43] Markus Swarowsky. Masterthesis code - Qin improvement attempt. https://doi.org/10.5281/zenodo.3724819, March 2020.

[44] Markus Swarowsky. Masterthesis code - reduce search space. https://doi.org/10.5281/zenodo.3731679, March 2020.

[45] Marcel Tiepelt. Quantum attacks on mersenne number cryptosystems. mathesis, Karlsruhe Institute of Technology, 2018.